AIRCRAFT COURSE OPTIMIZATION TOOL USING GPOPS MATLAB CODE

THESIS

Ryan D. Gauntt, Second Lieutenant, USAF

AFIT/GSE/ENV/12-M03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GSE/ENV/12-M03

AIRCRAFT COURSE OPTIMIZATION TOOL USING GPOPS MATLAB CODE

THESIS

Presented to the Faculty

Department of System Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Astronautical Engineering

Ryan D. Gauntt, B.S.

Second Lieutenant, USAF

March 2012

AFIT/GSE/ENV/12-M03

AIRCRAFT COURSE OPTIMIZATION TOOL USING GPOPS MATLAB CODE

Ryan D. Gauntt, B.S.

Second Lieutenant, USAF

Approved:

Dr. D. Jacques (Chairman)	date
Dr. R. Cobb (Member)	date
Lt. Col. F. G. Harmon, PhD (Member)	date

AFIT/GSE/ENV/12-M03

Abstract

Aircraft course planning between two points in a clear, no threat environment is easy and straightforward. However, the addition of various threats can greatly increase the difficulty and complexity of course planning. Placing new waypoints along the edge of each threat, mostly skirting the dangerous environment, may not prove too difficult, but such courses are far from optimal. Given aircraft, environment, and time constraints it is likely a much more optimal path exists between any given start and end points. This research focuses on determining the feasibility of using the *General Pseudospectral Optimization Software* program files written for the MATLAB® software package to take a given path, optimize it for the environment, and output a flyable, optimized course that can be used for more detailed mission planning. The results showed creating such a code was feasible. GPOPS can handle a simple version of what could be a very complex optimization problem. Two different versions of the final code show the successful optimization of the problem when the model is kept simple, and the failures GPOPS experiences when the problem becomes too complex.

Acknowledgements

This thesis would never have come to fruition without the help of those around me. I must first thank my friend 2Lt Robert Steigerwald, without who's math and MATLAB® skills, not to mention willingness to listen to my occasional diatribes, this paper might never have been finished. Also I must thank 2Lt Sam Barbaro, who, thanks to his hard work and dedication towards his own thesis, help keep my stress level in check. Thanks also must go to my primary advisor Dr. Jacques, who not only took me on as a thesis student but as an independent study student as well, in order to bring me up to speed on the concepts I would need in this research.

Finally, a major outpouring of gratitude must go to Cheri Adams, Jason Heaton, Kimberly Gresham and the entire staff at the Boonshoft Museum of Discovery. From the Cosmos to Legos, they accepted me as one of their own and provided me a place to go where wonder is still alive.

Ryan D. Gauntt

Table of Contents

Page

List of Figures

List of Tables

List of Symbols

List of Abbreviations

AIRCRAFT COURSE OPTIMIZATION TOOL USING GPOPS MATLAB CODE

I. Introduction

The art of aircraft course planning has existed since the first aircraft took flight. While planning for those first flights focused more on safely getting up and down alive, today's military aircraft must contend with a myriad of issues ranging from politically motivated "no-fly zones" to advanced radar arrays to simple shoulder mounted RPGs. Designing a course that safely navigates the dangers of a combat zone, while still quickly and efficiently fulfilling mission objectives, is an imperative that can quickly become a major bottleneck for mission planners. The ability to take a rough idea for an aircraft course and quickly optimize it would improve mission planning times greatly.

This quickly generated optimized path is not meant as a perfect flightpath. It may not, in reality, be truly optimal - it is only mathematically optimal for the information provided to the system that created it, and what is best in the mathematical realm can greatly differ from the real world. Far more detailed analyses, not to mention the input of human mission planners, would be required for any quickly generated path described here to be deemed acceptable. This does not mean computer code to find this optimum path is useless in the real world, it just means that what it outputs is only a starting point.

The objective of this research is to test the feasibility, usability, and effectiveness of a *General Pseudospectral Optimization Software* (GPOPS) based aircraft course optimizer. To that end, the Aircraft Course Optimization Tool (ACOT) was created. Composed of MATLAB® scripts and revolving around GPOPS, ACOT creates the framework defining the problem in a way GPOPS can use to find a minimum cost course for a given set of aircraft characteristics, threat conditions, and environment constraints. The goal is to produce an output that matches the needs discussed above with a reasonable degree of realism, including the inclusion of multiple radars, different power radars, simplified aircraft dynamics, and a nominal aircraft radar cross sections. While this output may not necessarily be of the highest fidelity, it is hopefully produced much more quickly by ACOT than by a person sitting at a table.

The next several chapters will discuss ACOT, its foundations, and its result in a way that explains its purpose and results. While nothing exactly like ACOT has been done before, there are other research areas which addressed similar issues and created solutions that run parallel to the same area as ACOT, but differ in various ways.

The majority of the thesis work focuses on ACOT itself since its creation was tantamount to determining the problem's feasibility. To start with, the large majority of the calculations take place inside the GPOPS scripts, created by Dr. Anil V. Rao at the *University of Florida*, and Standford University's Sparse Nonlinear OPTimizer(SNOPT) solver. The addition of several ACOT specific scripts frame the problem to the GPOPS code in the format it requires for all optimization calculations. These ACOT scripts set the optimization's differential equations of motion, its cost function, and the initial conditions to begin the optimization. Around these ACOT specific scripts a large shell code acts as a intermediary between the inner workings of ACOT and any end user, or more specifically a Graphical User Interface (GUI) with which the end-user interacts. That GUI, however, is not directly part of this thesis and all work was done through the shell code. As a whole, these scripts take in some basic information about the aircraft flying the course, the initial and final positions of the aircraft, the position and relevant strength information of any radars in the operating area, and a user provided initial path guess. This information is used inside GPOPS to find the minimum cost of a course based primarily on the amount of exposure to radars, but also the dynamic forces imposed by aircraft maneuvers, the mission's time of flight, and any no-fly/exclusion zones that may exist around specific areas.

Following a discussion of how ACOT works, the thesis looks at several different runs of the software to demonstrate some of ACOT's capabilities and limitations. As mentioned before, the output goal is meant to be a "quick and dirty" solution from which to begin a more detailed analysis. The output, while technically flyable by an aircraft, cannot be called mission ready.

II. Background

There are two major components to understanding the Aircraft Course Optimization Tool. The first part is a somewhat detailed grasp of what the tool, this thesis, is meant to accomplish. The second is a basic understanding of General Pseudospectral Optimization Software. Before those however, it is important to understand the general trajectory optimization problem. Several papers have discussed this problem, and many have solved problems similar to the Aircraft Course Optimization Tool, but none have directly discussed ACOT's specific topic.

2.1 Trajectory Optimization Studies

Work on aircraft course optimization has a long history. Over the past decade several people have looked at different models and methods for solving various problems of this type. A recent development in this area has been the use of systems like GPOPS, but other numerical and analytic solutions methods have been looked at before.

Michael Novy's thesis, *Air Vehicle Optimal Trajectories for Minimization of Radar Exposure*, looked at several of the underlying ideas behind ACOT. More specifically, Novy's thesis analyzed the various methods of solving radar exposure problems and compared some of the possible outcomes they produced. The dynamic nature of ACOT prevent the use of the analytic techniques Novy discussed, but his overview of optimal control directly corresponds to the method ACOT uses in its solution [3].

One interesting topic Novy discusses is the use of Voronoi diagrams in the implementation of optimal path planning through radar fields. In the context of radar fields, Voronoi paths are a geometrical method of finding the best path through a given field, assuming the location and powers of all radars are fully known. Furthermore, assuming all radars are of equal power, a Voronoi path will lie along the edges of geometrics shapes whos sides are all equidistant from neighboring radars. In the case of two radars, for example, the Voronoi path will be a perpendicular bisector to a line connecting the positions of the two radars [1, 3].

Jeffery Herbert, in his dissertation *Air Vehicle Path Planning*, also addresses the Voronoi path idea and provides an excellent image, reproduced below in Figure 2.1, that demonstrates the geometric shapes discussed above [1].

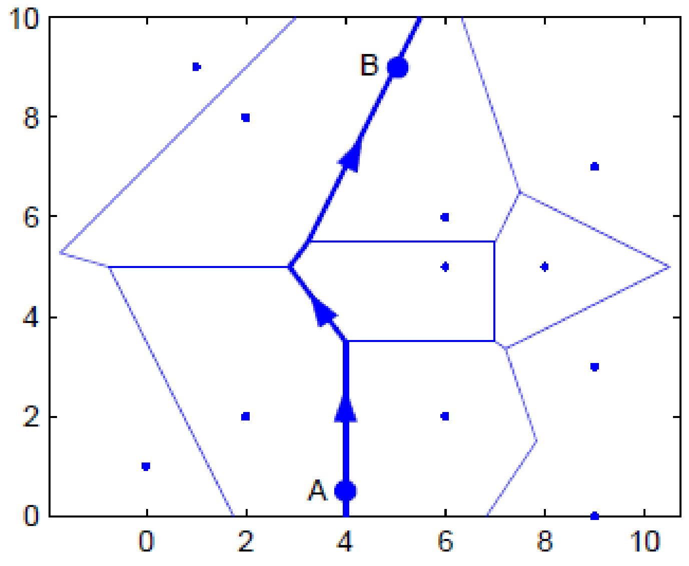

Figure 2.1: Voronoi diagram example from Herbert Dissertation [1]

Herbert also discusses, and then greatly expands on, the topics Novy discussed in his thesis. Not only does Herbert mention single vehicle path planning for single and multiple radars, but he analyses multiple vehicle control, a topic of great interest to ACOT but which at the current time has proven infeasible due to coding limitations. As it currently stands, the topics discussed by Herbert provide an excellent roadmap for ACOT and programs like it to follow. However, while ACOT is still successful in the problem it is trying to solve, it is unable to reach the levels of detail and open-ended nature of the problems discussed by Herbert [1].

One major source of information during early work on ACOT came from *Hybrid Gauss Pseudospectral and Generalized Polynomial Chaos Algorithm to Solve Stochastic Trajectory Optimization Problems* by Gerald Cottrill [4]. Discussing several different methods of optimization, and applying them to a basic trajectory optimization problem, Cottrill provided a framework defining the general problem ACOT should solve. In the paper, Cottrill defined a 2-D problem where an aircraft attempts to move from an initial point to another in an area with potential threats. The problem's objective was to find a path through those risks while minimizing the probability of the aircraft being killed by the previously mentioned threats. Furthermore, the threats themselves were assumed to have had a confirmed location at some point in time, but since then may have changed, e.g. an interceptor aircraft on patrol. This creates a probability distribution the aircraft must navigate through.

While ACOT eventually incorporated a more complex set of dynamics, early work focused on Cottrill's dynamics as seen in Equations 2.1, 2.2, 2.3.

$$\dot{x_1} = V\cos(\theta(t)) \tag{2.1}$$

$$\dot{x_2} = V\sin(\theta(t)) \tag{2.2}$$

$$\dot{\theta} = u(t) \tag{2.3}$$

where the vector $\bar{x}$ represents the x and y states (x_1 and x_2), V the velocity, and u the control variable that the optimization algorithm can modify to find the optimum solution. These dynamics required a constant speed assumption for the aircraft velocity and a control variable bound by the relationship

$$|u| = \frac{V}{R_{min}} \tag{2.4}$$

where R_{min} is a predefined minimum turn radius [4].

The optimization requires a cost function to operate. Cottrill's cost function, of the basic form seen in Equation 2.5, uses the probability distribution produced by the uncertainty of threat positions and the time of flight, to determine the flightpath's cost. Once provided to an optimizer, these cost functions are used with the dynamics to find the flightpath that minimizes the aircraft's exposure to threats [4].

$$J = \phi\left(x\left(t_f\right)\right) + \int_{t_f}^{t_0} L(\bar{x}, \bar{u}, t)dt \tag{2.5}$$

where $\phi\left(x\left(t_f\right)\right)$ determines the cost value for the final time state and $L(\bar{x}, \bar{u}, t)$ calcuates the cost of the continious portion of the optimization based on the state and control.

Another paper related to the research herein Timothy Jorris and Richard Cobb's paper *2-D Trajectory Optimization Satisfying Waypoints and No-Fly Zone Constraints* [5]. A similar concept to Cottrill's paper, this paper focused on a specific problem rather than various ways to solve a simple, general problem. As the paper explains, the Global Strike mission requires getting a hypersonic aircraft to a target in a minimum time while hitting required waypoints and avoiding prohibited areas. Clearly an optimization problem, Jorris goes through the development of the dynamics and cost function required for any optimizer. While in theory ACOT could be used to solve a problem such as this, Jorris' dynamics are normalized to values that match a hypersonic vehicle's operating regime, including both high speed and extreme altitude [5].

By analyzing Jorris' dynamics and combining them with the concepts discussed in *Annex A, Section 3 of Risk Management Plan for the Fleeting Target Technology Demonstrator* ACOT's dynamics developed their current form. This last paper discussed how an unmanned aerial system (UAS) with a fixed sensor could keep a target in its line of sight. In its discussion, the paper developed concepts which proved useful when creating ACOT's dynamics, such as assuming coordinated turns and using that assumption to relate load factor to bank angle and the increased load factor translates directly to a tighter turn [6].

2.2 General Pseduospectral Optimization Software

The General Pseduospectral Optimization Software created by Dr. Avil V. Rao from the University of Florida is a collection of MATLAB® scripts that work together to find the local minimum of a user provided cost function for a given set of differential equations. While GPOPS is at the heart of ACOT, its inner workings are not the focus of this research. Because of that this paper will not delve deeply into those working and will instead refer the reader to GPOPS specific documents [7].

Though not analysing GPOPS completely, it is prudent to describe the basics of General Pseduospectral methods and how GPOPS in particular uses them. First and foremost, GPOPS is a numerical solver. With Standford's Sparse Nonlinear Optimizer (SNOPT) at its core, GPOPS performs a gradient-based search to find the optimum solution to the problem provided to it. GPOPS and SNOPT iterate until the gradient search appears to have found a local minimum. Local minimum is used specifically here because the gradient-based method is not guaranteed to find the global minimum of a provided cost function, it can only find the closest minimum to the provided initial conditions [8].

The actual process is, of course, much more complex than described in the preceding paragraph and in reality relies heavily on the pseduospectral portion of GPOPS' name. More specifically GPOPS uses the *Radau Pseudospectral Method* that places additional collocation points at the best locations along the path to improve the quality and speed of the optimization. A detailed description of this method is beyond the scope of this thesis, but further imformation is avaliable in the GPOPS and SNOPT manuals. Suffice it to say, GPOPS uses internal algorithms to determine what points along a path it should focus on in order to best discretize the problem [8,9].

The final GPOPS code is a compilation of several dozen different MATLAB® scripts. The user, however, does not interact with the majority of these scripts but instead writes three to four scripts that define the problem in a way GPOPS, and thus SNOPT, understands. The first required script is the *main* script that sets up the initial conditions, limits, and first guess for GPOPS to start the problem with. The second is a script containing the problem's differential algebraic equations (dae), commonly referred to as the *dae* script. For problems such as those solved in ACOT, these are the differential equations of motion. In linear problems they are of the standard form

$$\dot{\overline{x}} = A\overline{x} + B\overline{u} \tag{2.6}$$

but, as can be inferred from the descriptions of GPOPS and SNOPT above, these differential equations need not be linear and instead take the form:

$$\dot{\bar{x}} = f(\bar{x}, \bar{u}, t) \tag{2.7}$$

Furthermore, the *dae* script also contains the calculations for any path constrains placed upon the problem - the values for which are defined in the *Main* script. These path constraints define areas in the solution space that the solution can not take.

The third required script contains the cost function of the problem, the set of equations that make GPOPS the optimization software it is. The cost functions contained inside are what GPOPS attempts to minimize during its run, typically of the form show in Equation 2.5 above and 2.8 here.

$$J = \phi\left(x\left(t_f\right)\right) + \int_{t_f}^{t_0} L(\bar{x}, \bar{u}, t)dt \tag{2.8}$$

where ϕ is the final state cost and L is the Lagrangian cost for the entire optimization problem. This cost function is calculated subject to the constraints and dynamics specficed as part of the *dae* script. The explicit time variant component, t, in Equations 2.7 and 2.8 are not required, and are not used in ACOT as this aircraft course optimization problem is kept as a nonlinear, time invariant problem that is only implicitly a function of time.

It must be stated the purpose of this thesis is not to analyze and discuss optimization and the math behind it, but instead to focus on a specific optimization problem. To that end, details about GPOPS and optimization can be found in referenced documents [4, 7, 8].

2.3 Aircraft Course Optimization Tool

The previous sections and their ideas combine with the complexity of mission planning to produce the problem ACOT attempts to solve. As the battlefield continues to change with the advent of new technologies it is becoming more and more difficult for people alone to properly take into consideration all those factors, at least in a reasonable amount of time. The Aircraft Course Optimization Tool is meant to ease the load on human mission planners by allowing them to input some basic information about the aircraft they are planing for and the threat environment into which the aircraft is heading and get an optimized solution as the output.

As with Jorri's paper, mentioned in Chapter II, the threats ACOT focuses on are those from stationary radar sites, though technically any circular threat area could be created and deemed a no-fly or high risk area. By looking at the properties of the aircraft, the given threats, and any provided time constraints, ACOT calculates the lowest threat path that meets all constraints.

The output from ACOT is not meant as a final product which can be given to pilots as part of their mission brief. Instead the output is created with further analysis in mind. So while an aircraft could fly the path ACOT creates, the output is meant to be used in conjunction with other threat analysis software. This higher fidelity software is much better able to analyze just how much of a threat the aircraft will encounter for all possible threats, not just the radars or other arbitrary threat rings ACOT considers. To do its analysis this high fidelity software requires a course to analyze. That is the course ACOT creates. The bottom line is that ACOT does not produce an end product, but creates something that is a stepping stone along the way. It is meant to aid in the work of human mission planners and their advance tools, not replace them.

III. Methodology

3.1 Early Development

The goal of this thesis is to determine the feasibility of using GPOPS, and thus the built in SNOPT, to determine the optimum course for an aircraft around various threats. The earliest development towards that goal revolved around properly framing the problem. In its most basic form, the problem was a standard optimization issue in which a cost function would be minimized subject to various constraints. As ACOT grew, so did the number and types of constraints. The cost function also changed with time, starting simple and eventually growing to something much more complex. In the end, this approach produced to distinct versions of ACOT, a Three Component State Vector (TCSV) version and a (FCSV) version. Sections 3.2 and 3.3 below talk about the development of both, while Chapter IV delves into what results they produce.

3.2 Three Component State Vector

The earliest working versions of ACOT consisted of a simple three component state vector, Equation 3.1, and their associated dynamics, Section 3.2.1. At first the problem was left without a cost function, instead relying on constrained no-fly zones. Once those simple problems demonstrated the code and dynamics worked, a cost function was developed and added to the problem, Section 3.2.2.

3.2.1 TCSV Dynamics. The TCSV dynamics are taken directly from Cottrill's work in *Hybrid Gauss Pseduospectrol and Generalized Polynomial Chaos Algorithm to Solve Stochastic Trajectory Optimization Problems* [4]. Both his state vector and dynamics are shown below in Equations 3.1, 3.2, 3.3, and 3.4.

$$\bar{\eta} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{3.1}$$

in which x is the x-position, y is the y-position, and θ is the aircraft heading. In ACOT's case ,the x-axis is also referenced as East, while the y-axis is North, since all plots are shown in the navigation frame used by pilots and navigators.

The TCSV dynamics, 3.2, 3.3, and 3.4, rely on simple planar geometry.

$$\dot{\eta}_1 = V\cos(\eta_3) \tag{3.2}$$

$$\dot{\eta}_2 = V\sin(\eta_3) \tag{3.3}$$

$$\dot{\eta}_3 = u(t) \tag{3.4}$$

In the dynamics V is the user set constant velocity while u is the control variable controlled by GPOPS and SNOPT. In the case of the three state dynamics the control vector is bound by the value calculated from Equation 3.5 below.

$$u_{bounds} = \pm\frac{V}{R_{turn}} \tag{3.5}$$

where V is the user defined constant velocity and R_{turn} the set turn radius.

For the simple TCSV model, the constant velocity and constant turn radius are two of the three aircraft properties the user must define. The third is the circular radar cross section that is used in the radar equation, Equation 3.8, in Section 3.2.2 below.

While not technically part of the dynamics equations, GPOPS considers both the dynamics and hard constraints at the same time during its optimization run. The only active path constraints in the middle of the course are those defining up exclusion zones around radars or any other point on the map the aircraft must completely avoid. To do that, the code uses Equation 3.6 to calculate the distance from the aircraft to the center of each exclusion zone and checks that value with the hard constraint. If the distance is less than the hard constraint, then the course is infeasible.

$$R^2 = (x - x_{R_n})^2 + (y - y_{R_n})^2 \tag{3.6}$$

3.2.2 TCSV Cost. The cost function for the TCSV model consists of three parts. The first part is the final time, which is used in the Mayer component of the cost function. The final two parts compose the Lagrange cost and calculate the costs incurred due to radars and any control usage.

The Mayer takes the simple form in Equation 3.7 where J_M represents the Mayer function value and t_f is the time it took the aircraft to fly the optimized course.

$$J_M = 10t^f \tag{3.7}$$

The 10 multiple weight on the cost was chosen because it worked well for the simple example problems used to test the code. It is expected that better values for this weight exist, but it is up to any end user as to what weights are needed for their specific problems.

The radar cost is based on the two way radar equation, Equation 3.8.

$$SNR = \frac{P_D}{P_R} = \frac{P_S G_S \sigma_T G_M \lambda^2}{(4\pi)^3 R^4 (kTB_M) L} \tag{3.8}$$

The variables for Equation 3.8 are defined in Table 3.1 [2].

As is clear both in Equation 3.8 and Table 3.1 this equation can prove difficult due to the multitude units that go into it. To simplify the problem the cost function in ACOT uses decibels, which not only makes all units equivalent but also switches the equation from multiplication and division to addition and subtraction.

$$SNR = P_s + G_s + G_m + \sigma_T + \lambda - kTBn - L - R - 33 \tag{3.9}$$

Table 3.1: Two Way Radar Equation Variables [2]

Type	Units	Description
$\widetilde{SNR}$	dB	Signal to Noise ratio limit that will pose a reasonable chance of aircraft detection.
P_s	watts	Power (average) transmitted from source.
G_s	dB	Gain of the source antenna.
G_m	dB	Gain of the receiving antenna.
λ	m	Wavelength of transmitted power.
L	dB	Loss factor.
σ_T	m^2	Radar Cross Section of reflecting object.
$kTBn$	dB	A measure of the internal noise power of the receiver, basically loss due to noise from the receiver temperature.
R^4	NM^4	Range from target to radar, raised to the fourth power due to inverse squared law acting twice on the two way travel.

where R is the decibel equivalent of R^4 and -33 is the decibel version of $(4\pi)^3$. For the TCSV model the radar cross section is not dependent on the orientation of the aircraft towards the radar. Regardless of the orientation the value is always the user defined circular radar cross section value.

Equation 3.9 is the decibel equivalent of Equation 3.8 and is the first step in finding the cost incurred through radar exposure. Initially the SNR calculated from Equation 3.9 was the cost, but it quickly became apparent that the negative nature of decibels when their non-decibel counterpart was a decimal ended up canceling out any incurred cost. To alleviate this the SNR from Equation 3.9 is normalized around a user defined detection-limit SNR and converted out of decibel form, Equation 3.10.

$$J_R = 10^{\frac{SNR - \widetilde{SNR}}{10}} \tag{3.10}$$

where J_R is the radar cost and $\widetilde{SNR}$ is the user defined detection threshold SNR. This forces the cost function to equal 1 when the aircraft is at the radar detection limit and quickly grow as it gets deeper into the threat area. When the aircraft is far away the cost is always less than 1 unless the cost from multiple radars adds up to create a higher overall cost.

For three equal power radars randomly distributed in a small area Equation 3.10 produces the surface in Figure 3.1, though in the figure the plot is kept in decibels since otherwise the scale would be impossible to read. Additionally, the top of the cost is shown as truncated, but once again this is for ease of display; when the cost is actually calculated there is no truncation to a maximum value.

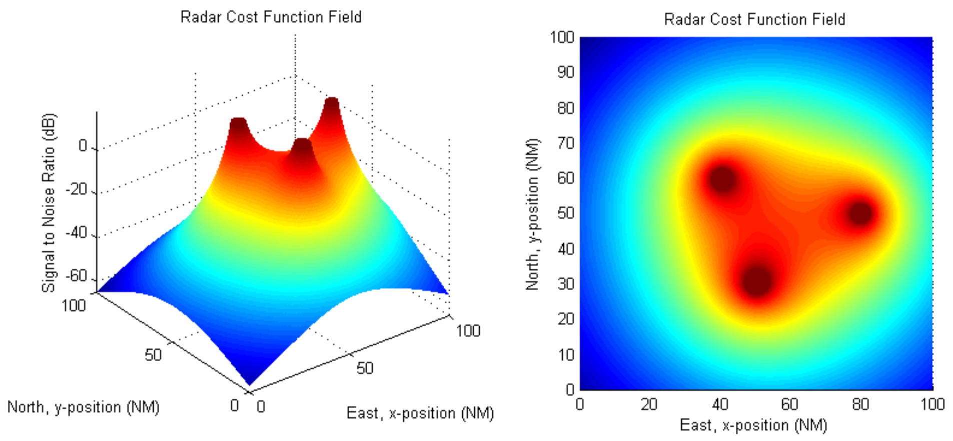

Figure 3.1: Example of a cost function associate with three equal power radars in decibels.

The last component of the cost function is the control cost. While this cost can be used to greatly reduce the amount of changes the optimizer will make to the path, its primary purpose here is to prevent the the optimizer from jumping back and forth over an optimum solution as the it struggles to get to the exact minimum when a few hundred thousandths off would not matter. In other words, adding the control cost speeds up the run time. For the TCSV this takes the form:

$$J_u = 10^{-4} u^2 \tag{3.11}$$

where the cost is simply a small fraction of the control's magnitude. The 10^{-4} weight on the cost was chosen because it worked well for the simple example problems used to test the code. As with the weight on the Mayer cost, it is expected better values for this weight exist, but once again it is up to any end user as to what weights are needed for their specific problems. This is also true for any weights on the radar cost, though for the current cost here the weight is left as 1.

The final cost is calculated by summing the Mayer cost with the integral of the Lagrange cost, Equation 3.12.

$$J = 10t_f + \int_0^{t_f} J_R(t) + 10^{-4}u(t)^2 dt \tag{3.12}$$

where both the range term in the J_R radar cost calculation and the control variable, u, vary with time.

3.3 Five Component State Vector

It quickly became clear that as ACOT's development continued, the TCSV dynamics, while they worked, failed to capture the problem in as much detail as initially hoped. They performed well, and technically proved that it is feasible to create an aircraft course optimization tool using GPOPS, but something more was need. Thus work began on the five component state vector model.

3.3.1 FCSV Dynamics. The five component state vector model is an extension of the three component model. Three of its five states are direct copies of the state in the TCSV, but how their dynamics are calculated is modified to take more conditions into account. The five states, x-position, x; y-position, y; velocity, ν; heading, θ; and bank angle,ϕ are seen in Equation 3.13.

$$\bar{\eta} = \begin{bmatrix} x \\ y \\ \nu \\ \theta \\ \phi \end{bmatrix} \tag{3.13}$$

For this more advanced model the velocity is no longer constrained and is instead allowed to vary, while the heading derivative is based on more than just a single control parameter. With that said, the first three differential equations come quite easily based on simply planar geometry or a control definition.

$$\dot{x} = \nu \cos(\theta) \tag{3.14}$$

$$\dot{y} = \nu \sin(\theta) \tag{3.15}$$

$$\dot{v} = U_1 \tag{3.16}$$

Equation 3.16 show that U_1, the first component of the two component control vector $\bar{U}$, is the acceleration of the aircraft, and that the optimizer has direct control over the aircraft velocity.

After the simplicity of the first three differential equations, the last two take a bit more derivation. To begin, Figure 3.2 shows an aircraft free body diagram with several forces labeled. This diagram is used extensively to derive the last two differential equations. The terminology for Figure 3.2 is defined in Table 3.2.

Mentioned before, ACOT is a two dimensional problem. The underlying assumption here is that the vertical component of lift, L_v, is always equal to the aircraft weight, W, so that there is no vertical acceleration. This assumption allows for the

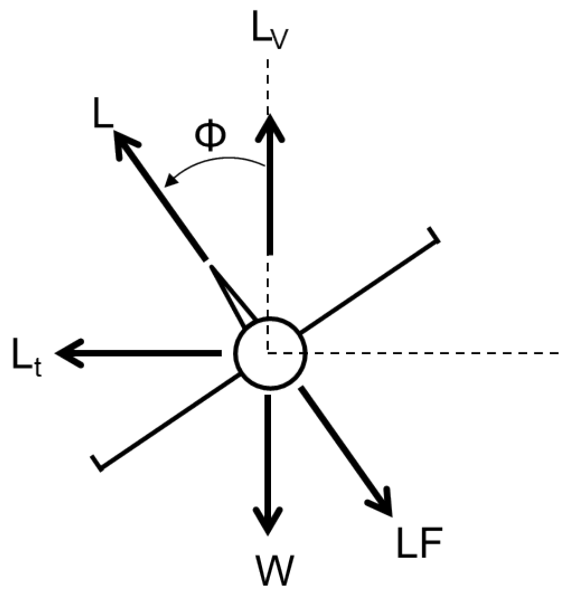

Figure 3.2: Aircraft Free Body Diagram

Table 3.2: Aircraft FBD Description

L	Lift produce by wings
W	Weight of aircraft
L_v	Vertical component of lift
L_t	Component of lift that causes turn; centripetal component
LF	Load Factor
ϕ	Bank angle

final derivation to continue, beginning with the basic force equation seen in Equation 3.17.

$$F = ma \tag{3.17}$$

it follows that

$$L_t = L\sin(\phi) = ma_t \tag{3.18}$$

$$a_t = \frac{L}{m}\sin(\phi) \tag{3.19}$$

where F is Force, m aircraft mass, and a acceleration. This acceleration is not the same variable as the acceleration mentioned as the first control variable in Equation

3.16, as it does not refer to acceleration in the direction of aircraft travel, but acceleration in the plane of the paper. Here it specifically relates to turning maneuvers, though more specifically the t subscript denotes turn inducing forces or accelerations. Since the $\sin(\phi)$ multiplier in a_t finds the turn inducing component of acceleration, the term L/m is the total acceleration experienced by the aircraft. As the amount of lift is unknown, it is not possible to directly solve for this acceleration, but by assumming the vertical component of lift is always equal to the weight of the aircraft it is possible to remove the lift and weight terms from the problem completely. Starting with

$$L = \frac{L_v}{\cos(\phi)} \tag{3.20}$$

and keeping in mind $L_v = W$,

$$L = \frac{W}{\cos(\phi)} \tag{3.21}$$

Rearranging to solve for $\frac{L}{W}$, Equation 3.21 becomes

$$\frac{L}{W} = \frac{1}{\cos(\phi)} \tag{3.22}$$

and since lift over weight equals the load factor, Equation 3.22 becomes

$$LF = \frac{1}{\cos(\phi)} \tag{3.23}$$

Equation 3.23 is not in usable units, but is instead in terms of G's. To compensate for this, ACOT multiplies the load factor by a Earth's gravitational acceleration in NM/hr^2 to convert from a G-load to a true acceleration (recall all time units in ACOT are in terms of hours).

Using the relationships in Equations 3.24, 3.25, and 3.26 it is possible to combine Equations 3.19 and 3.23 into Equation 3.27

$$W = mg \tag{3.24}$$

$$\frac{L}{m} = \frac{Lg}{W} \tag{3.25}$$

$$\frac{L}{W}g = LFg \tag{3.26}$$

$$a_t = LFg\sin(\phi) \tag{3.27}$$

To finish the derivation, it is necessary to return to basic circular motion. As seen in Figure 3.3

$$\bar{a_c} = \bar{\omega} \times \bar{V_t} \tag{3.28}$$

where $\bar{a_c}$ is centripetal acceleration, $\bar{\omega}$ angular velocity, and $\bar{V_t}$ tangential velocity (here is the only place where t subscript denotes tangential component).

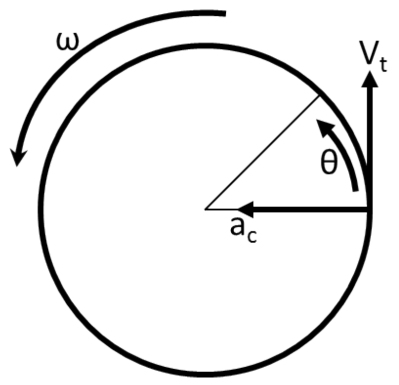

Figure 3.3: Basic Circular Motion

By looking at the vector magnitudes, Equation 3.28 is written as

$$|\bar{a_c}| = |\bar{\omega}| \left|\bar{V_t}\right| \sin(\theta) \tag{3.29}$$

where θ is the angle between the two vectors $\bar{\omega}$ and $\bar{V_t}$. In the case of circular motion $\theta = 90^\circ$, $\sin(\theta) = 1$ and, dropping the magnitude symbols, Equation 3.29 becomes

$$a_c = \omega V_t \tag{3.30}$$

The circular motion terms introduced in Equation 3.28 relate to the ACOT differential equations by the relationships in Equations 3.31, 3.32, 3.33.

$$a_t = a_c \tag{3.31}$$

$$\dot{\theta} = \omega \tag{3.32}$$

$$\nu = V_t \tag{3.33}$$

Thus, by combining these relationships with the previously derived equations,

$$\dot{\theta} = \frac{a_t}{\nu} \tag{3.34}$$

Rearranging and substituting 3.27 reveals

$$\dot{\theta} = \frac{LF\gamma \sin(\phi)}{\nu} \tag{3.35}$$

and

$$\dot{\phi} = U_2 \tag{3.36}$$

where, as with $\dot{\nu}$, U_2 is the second component of the control vector $\bar{U}$.

Putting these equations in terms of the state vector yields

$$\dot{\eta} = \begin{bmatrix} \eta_3 \cos(\phi) \\ \eta_3 \sin(\phi) \\ U_1 \\ \frac{LFg \sin(\eta_5)}{\eta_3} \\ U_2 \end{bmatrix} \tag{3.37}$$

with the control vector

$$\bar{U} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \tag{3.38}$$

As with the TCSV model, the FCSV code considers the active path constraints at the same time as it calculates the dynamics. The same equation as in the TCSV, Equation 3.6, calculates the distance from the aircraft to the center of each exclusion zone and checks that value with the user provided hard constraint. If the distance is less than the hard constraint, then the course is infeasible.

Finally, these complex dynamics require the user to provide more information about the aircraft used during the optimization, as the simple maximum speed and constant turn radius from the TCSV code does not fully define the problem. These new dynamics use variable speeds and calculate turn radius based on bank angle and speed. Table 3.3 defines the required aircraft information for this five component model of the problem.

3.3.2 FCSV Cost. In the same way the FCSV dynamics were an extension of the TCSV dynamics, so too are the FCSV costs an extension of the TCSV cost. The primary concern for both are the radar costs and the finial time, though in the FCSV version a few more components are added to increase the number of factors the optimizer considers.

The minimum time cost function for the FCSV, Equation 3.39, is the same as with the TCSV

$$J_M = 10t_f \tag{3.39}$$

Though technically possible with the TCSV model, the inclusion of a fixed final time designator is reserved for the FCSV model here. Equation 3.40, however does not make the problem a true fixed final time problem as it is a cost rather than constraint. While GPOPS can optimize for a fixed final time it was decided a better

Table 3.3: FCSV Required Aircraft Information

Type	Units	Description
Minimum Speed	kts	Minimum speed the aircraft is allowed to travel. Recommended this is far enough above banked turn stall speed that aircraft will not stall if turning while traveling this slowly.
Maximum Speed	kts	Maximum speed the aircraft will be allowed to travel.
Max Acceleration	kts/s	Maximum amount of acceleration allowed during optimization.
Max Deceleration	kts/s	Maximum amount of deceleration allowed during optimization.
Max Load Factor	g's	Maximum g-forces allowed. Used to calculate maximum bank angle, as ACOT assumes coordinated turns.
Max Bank Rate	deg/s	Fastest the aircraft is allowed to bank during optimization.
RCS File	m^2	A tabulated file, currently must be .mat, that contains the radar cross section for the aircraft. There are two columns, the first is the angle on the aircraft from nose in degrees in one degree increments, the second is the size of the cross section in m^2 . An example is contained in Appendix 1.

solution for this problem would be a goal final time with any deviation from that goal heavily penalized. This "goal" model allows for deviation from a fixed final time if the optimizer finds an extremely low cost path to make up for the time penalty.

$$J_M = e^{\left|t_f - \tilde{t_f}\right|} \tag{3.40}$$

where $\tilde{t_f}$ is the user defined desired final time.

The radar cost for the FCSV version of ACOT is the same as for the TCSV model.

$$SNR = P_s + G_s + G_m + \sigma_T + \lambda - kTBn - L - R - 33 \tag{3.41}$$

In the five component version, however, there is an additional time variant component to the radar cost: a variable radar cross section. In the TCSV code the aircraft is defined with a circular radar cross section who's value remained the same regardless of the aircraft's orientation to the radar. In this more complex version of the code the cross section is dependent on the aircraft heading. Instead of defining one value for the entire aircraft, the user instead provides a table with the cross section as a function of angle on the aircraft. Each time the optimizer calculates the cost of a path it determines the orientation of the aircraft and picks what the cross section it is presenting towards the radar.

As with the TCSV code, the SNR calculated by Equation 3.41 is normalized about the user defined detection limit SNR and converted out of decibels.

$$J_R = 10^{\frac{SNR - \widetilde{SNR}}{10}} \tag{3.42}$$

Finally, there is still a small cost component associated with the control vector. Once again, this cost is less to penalize control usage and more to prevent GPOPS from attempting to make changes that may produce mathematical results that, in the real world, would prove quite pointless. Additionally, this cost helps ensure GPOPS will not jump back and forth between two correct solutions that differ only in amount of control usage.

$$J_{\bar{U}} = 10^{-15}u_1^2 + 10^{-6}u_2^2 \tag{3.43}$$

As shown in Equation 3.43, the cost is built from the squared magnitude of the control values, multiplied by weights. As with the weights discussed earlier, these were chosen specifically for how they performed in the example problems used to test ACOT. Better values for them do exists, but those values are based on the specific problems a user needs to solve. The weights on the other components of this cost fall under this same descriptor. They work for the examples run to test ACOT but need to be set by a user for their specific problems.

As with the TCSV, the final cost is calculated by summing the Mayer cost with the integral of the Lagrange cost, Equation 3.44.

$$J = J_M + \int_0^{t_f} J_R(t) + 10^{-15}u_1(t)^2 + 10^{-6}u_2(t)^2 dt \tag{3.44}$$

3.4 Aircraft Course Optimization Tool Code

The Aircraft Course Optimization Tool is a set of MATLAB® scripts that define the problem in a way GPOPS can utilize. It is with these scripts the user interacts. Currently there are two versions of these script: one for the three component state vector and one for the five component state vector. While the two scripts do vary in their internal make up, they function the same way and thus the discussion below applies to both equally, unless otherwise stated. As Figure 3.4 shows, the code is designed so all user interaction takes place with the ACOT Shell. Once the user has provided the necessary information to the shell code the information is handed off to the Main Script. This scrip takes the prepared information and packages it into various structures that are required for GPOPS to function properly. When all the information is properly sorted, the Main Script calls the actual GPOPS code.

From there the code begins its optimization. The optimization itself takes place in SNOPT, as discussed in Chapter II, with GPOPS improving the process by adding various collocation points. While SNOPT and GPOPS are optimizing they reference the last two user defined scripts, the DAE Script and the Cost Script. The DAE script contains one of the sets of differential equations discussed in Sections 3.2.1 or 3.3.1 while the Cost script defines one of the cost functions discussed in Sections 3.2.2 or 3.3.2

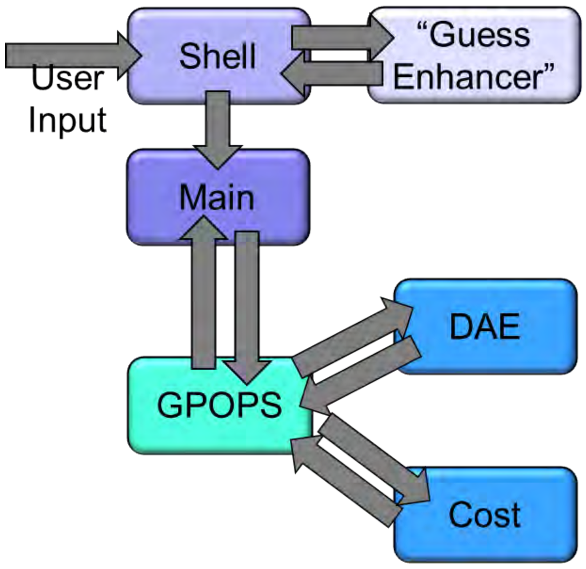

Figure 3.4: Flow of ACOT and GPOPS Code

3.4.1 ACOT Shell. As ACOT is currently designed, nearly all user interaction with the ACOT takes place within the *ACOT Shell* (MATLAB® file name courseOptShell.m). Users can, and should, modify the costs in the cost function script, but all problem input takes place within ACOT Shell. ACOT Shell is formatted in a more user-friendly manner than the standard GPOPS code, and while it can be used by a person, the shell was created with the understanding that it would interface with a GUI rather than directly with a user.

ACOT Shell has five main areas. The first section is where user sets the properties defining the aircraft model. Depending on the version of the code, either three state or five state, exactly what properties are needed will vary. Because the variables required for the aircraft model were discussed elsewhere in the paper, and are defined throughly in the code itself, a detailed discussion will be omitted here.

The second section of ACOT Shell contains the radar information. Here the user provides the locations, gains, powers, and detection limits of any radar in the problem. As with the aircraft information, a detailed description of these variables will be omitted here since they are discussed in the code and previous sections.

The initial and conditions of the aircraft are next. For both the three state and five state versions the initial and final positions are required. The FCSV code also requires the input of initial and final velocities and headings, though they are free to change unless the user sets them as constrained values.

Following the conditions is the initial guess. This vector provides a starting point for the optimization. It contains the initial and final positions, as well as any intermediate waypoints the user has determined will improve the optimized solution. These intermediate points are not hard constraints, the optimizer can freely pick a path off of these points. They exist to only to start the problem. Typically, the closer this initial guess is to an optimum solution the faster that solution will be found. As GPOPS and SNOPT find local minimums, the final optimized solution can vary greatly depending on where the initial guess is placed. The gradient search will start from there and move towards the closest local minimum.

The coordinate system ACOT uses for both user input and optimized solution output is the navigation frame typically used for aircraft navigation. Shown in Figure 3.5, this coordinate system has the x-axis pointing north and the y-axis east. While its not used for ACOT, the z-axis points down. Using the right-hand rule demonstrates that, with $0°$ along the x-axis, the heading increases in a clockwise direction, which matches any magnetic compass or aircraft heading indicator in the world. Thus any positions put into ACOT use nautical miles east by nautical miles north, while any headings start with $0°$ in the northern direction and increase as the aircraft turns East.

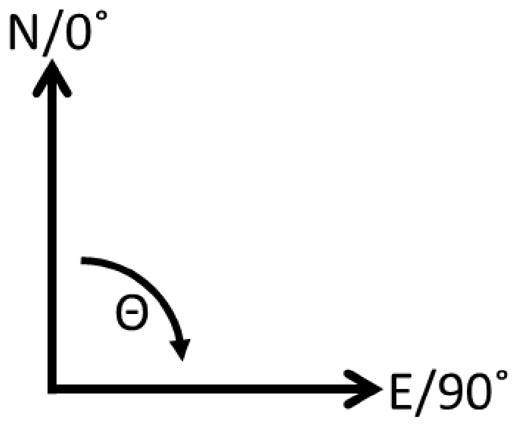

Figure 3.5: Navigation Frame

While the user only need provide positions for an initial guess, GPOPS requires every state and every control to be defined at every initial guess point. Rather than requiring the user to input every state, a tedious task for the FCSV model, the Shell Code calls on the "Guess Enhancer." The Guess Enhancer is a small script that fills in the empty components of the initial guess. For the TCSV it adds in the new aircraft heading at each initial guess point and sets the instantaneous heading change (the control) to zero. For the FCSV, the Guess Enhancer sets the velocity to the aircraft maximum, calculates the new aircraft heading, and sets the bank angle to maximum in the direction of heading change. The two control variables, velocity change and bank rate, are set to zero. While the maximum velocity and bank angle assignments may seem extreme, and the zero control unrealistic, GPOPS is not forced to stick with these values. As stated in the preceding paragraph, the optimizer can freely pick values off of this initial guess. It acts only as a starting place.

The last user input required in ACOT Shell is setting which plots should be presented before and after the run. For instance, before the optimization begins ACOT can display a plot of the radars and initial guess, providing the user with an opportunity to change the guess if it does not fit the problem. After the optimization is complete any of the states, as well as the controls, can be displayed for analysis.

In addition to user input and creating a fully defined initial guess, ACOT Shell begins the data preparation for GPOPS. It converts all units into the units used throughout ACOT and assigns any global variables and flags that are required during the optimization. When the data is fully prepared ACOT Shell calls the ACOT Main Script.

3.4.2 ACOT Main (courseOptMain.m). Shown in Figure 3.4, ACOT Shell hands off the problem to ACOT Main. Most problems that use GPOPS start with a main script that properly labels and sets up the variables and structures for the problem. When work began on ACOT, this main script is where everything began. The shell code was created to make the problem definition process a bit easier and to provide a clean basis from which to add a GUI. Regardless, there are several important functions the ACOT Main script performs. To begin with, the it acts as a buffer between ACOT Shell and GPOPS itself. The main script officially declares the initial conditions, final conditions, minimum and maximum values for all the states, be it the three or five component state vector problem, and places the guess vector and all other variables into various global structures. These global structures are then used through GPOPS and SNOPT for the optimization.

ACOT Main also defines the internal setting GPOPS uses during it's optimization. While these settings are of little consequence to the end user they can greatly change the outcome of the problem if changed. The manner in which they are currently set works well for ACOT's needs.

Once the global structures are filled and GPOPS' internal settings defined, ACOT Main calls the actual GPOPS code. From here the internal working of GPOPS and SNOPT take over. During its run, GPOPS calls on the ACOTS scripts defining the problem's dynamics and cost, both of which are described below.

When the optimization problem finishes, be it a successful or failed optimization, GPOPS outputs information back to ACOT Main that is used to create any plots the user requested back in ACOT Shell. In addition to the plots, GPOPS keeps all information stored in the MATLAB® workspace for use in further analysis.

3.4.3 ACOT Dynamics (courseOptDae.m). ACOT Dynamics contains the problem specific dynamics discussed in Sections 3.2.1 and 3.3.1 above. GPOPS uses this code to determine how the aircraft moves over the duration of the optimization. However, not only does this script contain the differential equations of motion, it is also where any path constraints are applied to the problem. In the case of ACOT the only path constraint in ACOT Dynamics are those calculating the radar exclusion and no-fly zones discussed in the dynamics sections above.

While the equations do not diverge from those discussed earlier, the form the equations take in the code are such that GPOPS is able to calculate the state derivative for every state, at every point in time it has created a collocation point. This is due to the large array GPOPS uses to store the states, where each column is a separate state and the rows are that states' value for each collocation point. The state derivatives are also stored in the same type array for use in updating the aforementioned states back in GPOPS.

3.4.4 ACOT Cost (courseOptCost.m). The last ACOT specific script is ACOT Cost. This code defines the cost function used during the optimization. As with the differential equations of motion, the contents of this script depends on whether the problem is the TCSV or FCSV version. In both cases the equations discussed in Sections 3.2.2 and 3.3.2 are slightly modified for the code in that the equations above are in scalar form, but in the cost function code they are written as linear algebraic equations. This occurs for the same reason the state derivatives were stored in large arrays: GPOPS calculates the cost at each collocation point before integrating them over the entire flight time. In the code, the Lagrange is a large vector created by summing the cost from the radar and control Lagrange components.

IV. Results

With an infinite number of possible starting locations, end points, radar positions and strengths, it is impossible to completely characterize ACOT's behavior within the time constraints of this investigation. With that said, however, it is withing the realm of this thesis to demonstrate the feasibility of using GPOPS for aircraft course optimization. To that end several examples from the three component state vector and five component state vectors versions of ACOT are discussed below. That discussion will include the initial and final conditions, aircraft information, radar information, and the optimized output for a each problem, if it is possible to discuss them. Furthermore, each successful example notes the amount of computer time each run required. Each example has been performed on the same type computer with nothing except MATLAB® version 2011a running. Additionally, the command to gauge the run time of the code looks at just the scripts involved in prepping and running GPOPS. The time spent in the shell code and creating any plots is not considered. For any examples which fail to run, or run with issues, an emphasis is place on discussing where the issues might lie.

4.1 Three Component State Vector Results

The first version of ACOT to truly work, the three component state vector code is the basis from which all runs are gauged. The examples below start with simple one radar problems to verify that the code works as expected and to demonstrate how GPOPS finds the closet local minimum of the cost function. After the single radar problem is successfully demonstrated, the complexity of the problem is increased to first three, and then finally seven radars. The cost function is verified as part of the three radar example.

4.1.1 TCSV Single Radar Problem. The easiest way to demonstrate the basic functionality of the ACOT code is with a simple single radar problem. This example, as well as all following examples, use the aircraft model shown in Table 4.1 and radar model in Table 4.2. Recall that all information given in the radar properties is eventually converted to decibels. For the single radar problems the radar is located at 30 NM East and 50 NM North.

Table 4.1: TCSV Aircraft Properties

Type	Units	Value
Ground Speed	kts	200
Turn Radius	NM	2
RCS	m^2	5

Table 4.2: Single Radar Problem Radar Information

Type	Units	Property Value
P_s	watts	40000
G_s	dB	40
G_m	dB	40
f	kHz	$9.9931E-6$
$kTBn$	dB	-120
L	dB	3
$\widetilde{SNR}$	dB	13
$KillSNR$	dB	18

With the aircraft and radars defined, only the initial guess is required to begin the optimization. For this first example the initial guess is seen in Table 4.3.

Table 4.3: TCSV Single Radar Problem Initial Guess

East (NM)	North (NM)
0	50
30	25
100	50

Table 4.3 shows the aircraft starting at 0 NM East, 50 NM North and heading south to 30 NM East, 25 NM North, or directly below the radar. At that point the aircraft turns back to the north and heads to its final position of 100 NM East, 50 NM North.

Running all these values through GPOPS produces the optimized course in Figure 4.2, with the cost and computer run time seen in Table 4.4. In the figures the red inner circles represents the no-fly or exclusion zone around the center of a radar that is calculated from the $killSNR$ value provided in the radar properties. The outer green circle is the $\widetilde{SNR}$ detection limit ring, while the solid blue line either the initial guess course or the optimized course (depending on the figure).

Table 4.4: TCSV Single Radar Problem Final Cost and Run Time

Cost	CPU Run Time (sec)
2.3324	3.15

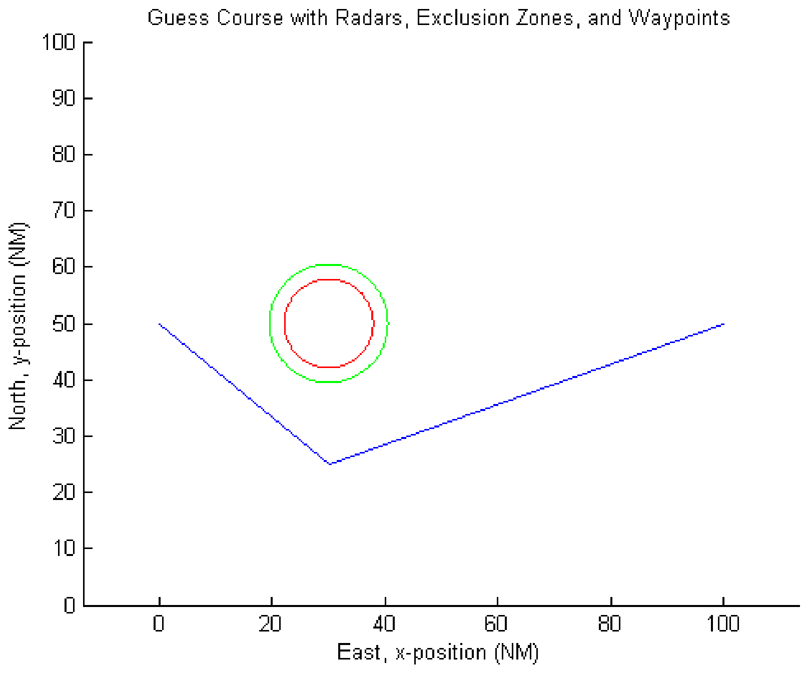

Figure 4.1: TCSV Single Radar Problem Guess Course

The quick run time and the optimized result in Figure 4.2 clearly demonstrates the success of this first example. In a short period of time the optimizer successfully found the best *southernly* course around the radar using the provided aircraft properties, radar properties, and initial guess.

One useful check to run on this results is to verify that the control vector output by GPOPS will, when run through the problem dynamics, produce the same result when using a numerical ODE solver like MATLAB's® built in ODE45. Running this test produces the outcome in Figure 4.3, in which the solid blue line is still the GPOPS

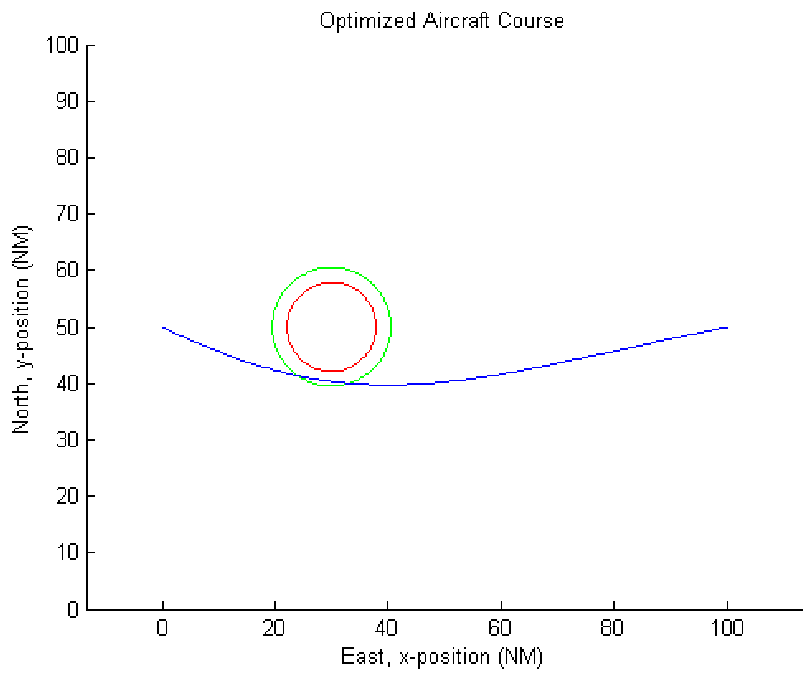

Figure 4.2: TCSV Single Radar Problem Optimized Course

optimized course and the dashed red optimized course is the ODE45 calculated course using the TCSV dynamics and GPOPS output control vector. The two are difficult to distinguish from each other as the ODE45 calculated path exactly matches the GPOPS path.

4.1.2 TCSV Single Radar Problem with Alternate Initial Guess. Changes in the initial guess can change the final optimized output. Discussed before, GPOPS searches for the closest local minimum to the initial guess. For this single radar problem, switching the initial guess from a southern path to northern causes the optimized path to do the same. Specifically, the initial guess changes the intermediate point from 30 NM East, 25 NM North to 30 NM East, 75 NM North; otherwise this example is the exact same as in Section 4.1.1.

As expected the optimized course in Figure 4.5 hugged the northern limb of the radar detection ring. Because this example is symmetrical to the previous one, the cost function ends up with the same value as before, 2.3324. If the problem was not

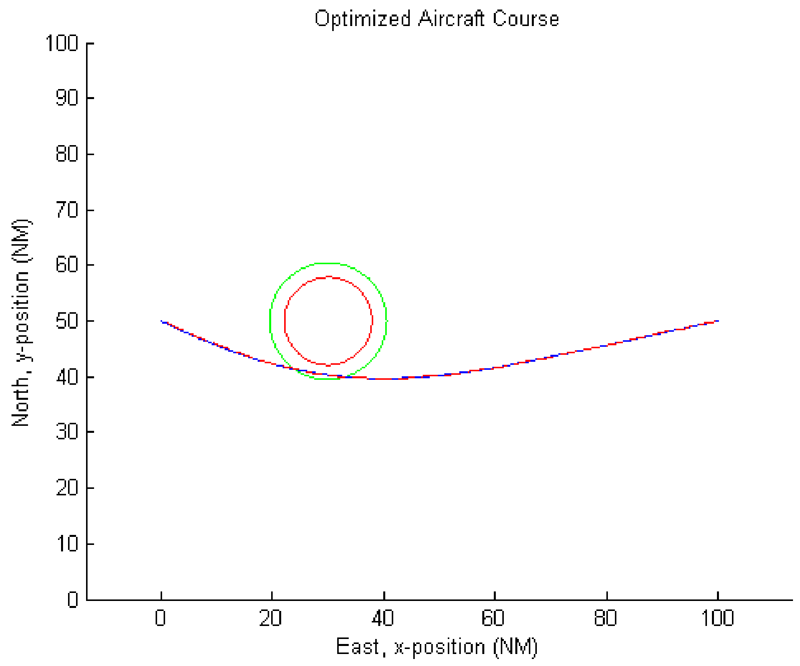

Figure 4.3: TCSV Single Radar Problem ODE Verification

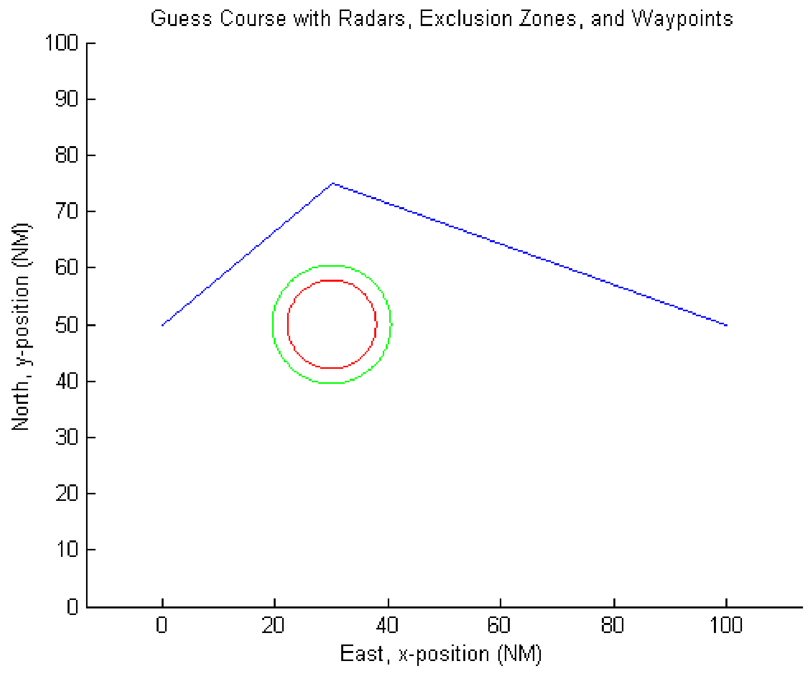

Figure 4.4: TCSV Single Radar Problem with Alternate Initial Guess Course

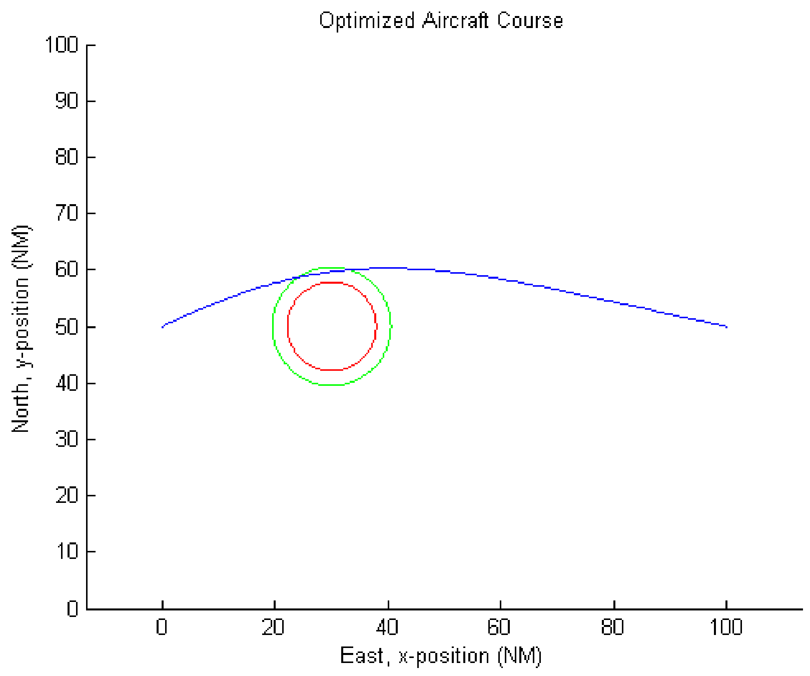

Figure 4.5: TCSV Single Radar Problem with Alternate Initial Guess Optimized Course

symmetrical, and the southern route had a lower cost, the code would not be able to reach it from the northern initial guess. The gradient always leads away from the radar, so there is no way the gradient based solver would have looked on the opposite side of the radar cost hump for a globally lower cost. While it was not necessarily true for this case, other situations rely much more heavily on a good initial guess close to the globally lowest cost path if that path is to be found.

4.1.3 TCSV Three Radar Problem. The first multiple radar problem contains the same radar as the single radar problem, but adds two additional radars at 60 NM East, 55 NM North and 50 NM East, 20 NM North. The initial guess for this problem contained one additional intermediate point. The entire guess is displayed in Table 4.5.

Figures 4.6 and 4.7 show both the initial guess from the above table and the optimized course. The run produced the cost and CPU run time results in Table 4.6

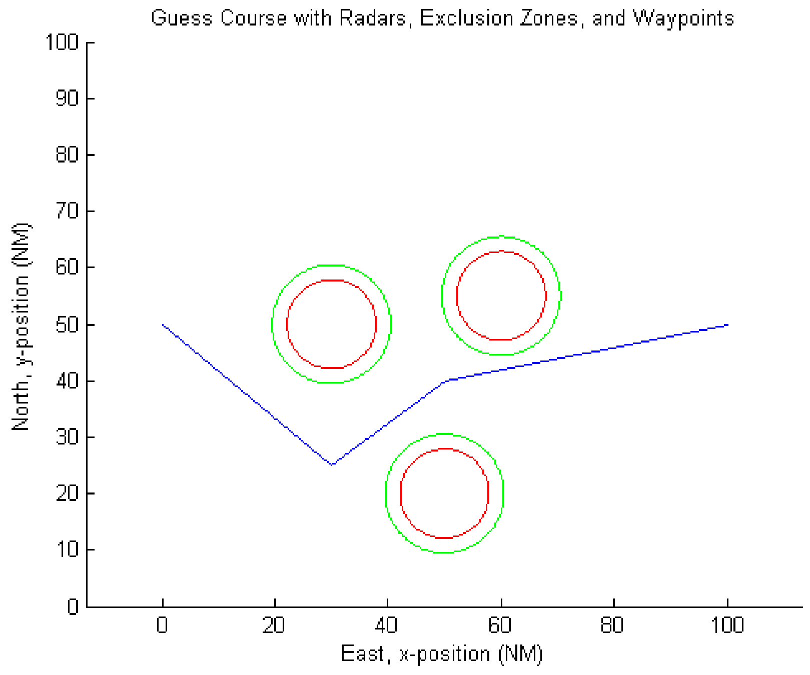

Figure 4.6: TCSV Three Radar Problem Guess Course

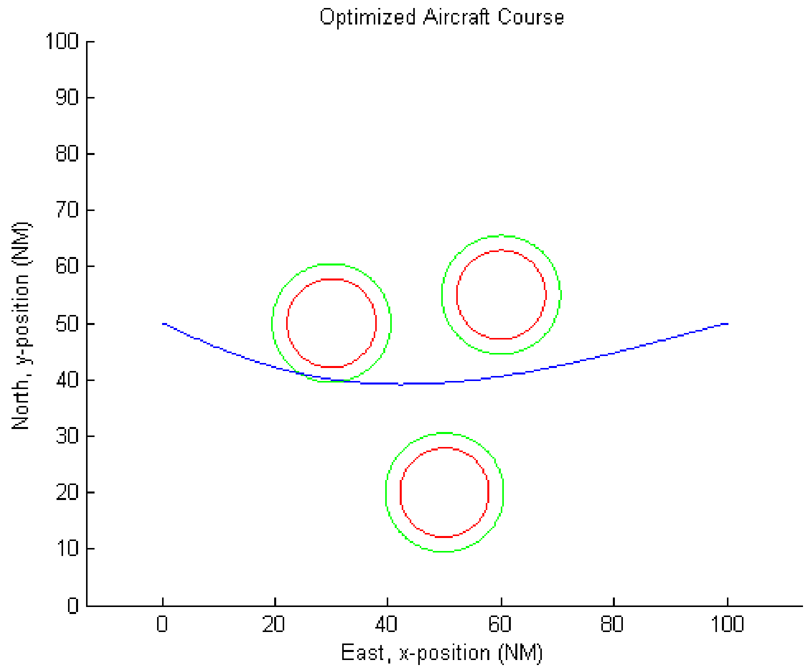

Figure 4.7: TCSV Three Radar Optimized Course

Table 4.5: TCSV Three Radar Problem Initial Guess

East (NM)	North (NM)
0	50
30	25
50	40
100	50

Table 4.6: TCSV Three Radar Problem Final Cost and Run Time

Cost	CPU Run Time (sec)
2.3553	3.81

Once again the successful optimization is clear. However the cost function itself has not been verified. While GPOPS and SNOPT claim that the problem as been optimized, further analysis is possible. By taking the output from this example and perturbing it slightly to both the north and south, then running all three courses through the cost function direction, created the three costs in Table 4.7 and the output plot in Figure 4.8.

Table 4.7: TCSV Cost Function Verification

GPOPS Output	2.362563
Perturbed North	2.368556
Perturbed South	2.368554

The higher costs associated with the non-optimal perturbed courses prove that the cost function is properly optimized by GPOPS. Furthermore, this mean the GPOPS output can be trusted to be the lowest cost possible from the provided initial guess.

4.1.3.1 TCSV Seven Radar Problem. The easy with which the TCSV code optimized the first two examples bring up the question of just how complex of a problem can ACOT handle, or specifically the TCSV version of ACOT. To test this several more radars were added to the problem. All seven of them, their locations shown in Table 4.8, are all equal power and match the radars used in Sections 4.1.1 and 4.1.3. The new radar positions also call for a new initial guess. This guess is

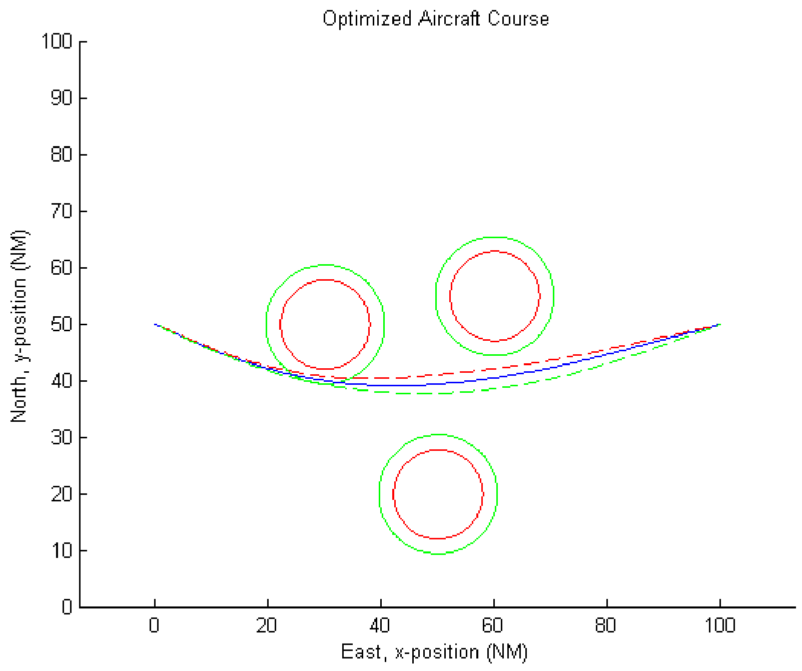

Figure 4.8: TCSV Cost Verification Courses

similar to the three radar problem in that it has two intermediate points, but the second point is located farther east and north. The full guess can be seen in Table 4.9

Table 4.8: TCSV Seven Radar Problem Radar Positions

Type	Units	Radar 1	2	3	4	5	6	7
EastPosition	NM	30	50	70	50	70	70	30
NorthPosition	NM	40	20	35	45	60	20	60

Running the optimization with these inputs produces the guess plot in Figure 4.9 and the optimized course in Figure 4.10. The total cost and CPU run time are in Table 4.10. Even with seven radars, GPOPS and SNOPT are able to quickly and easily optimize the aircraft course. The output, while not mission ready, is clearly flyable.

4.1.4 Three Component State Vector Comments. While every one of the above examples performed well, there are still a few items to discuss in how the

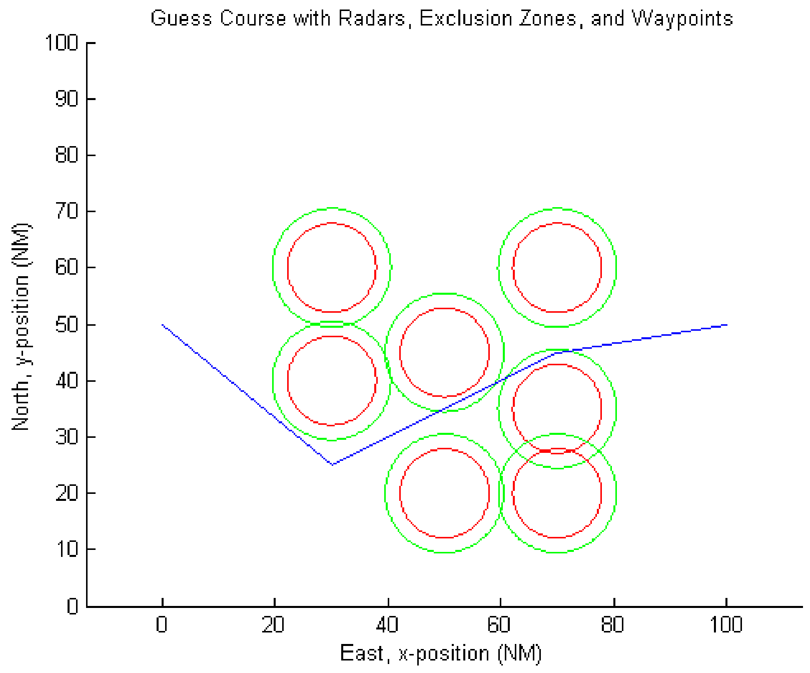

Figure 4.9: TCSV Seven Radar Problem Guess Course

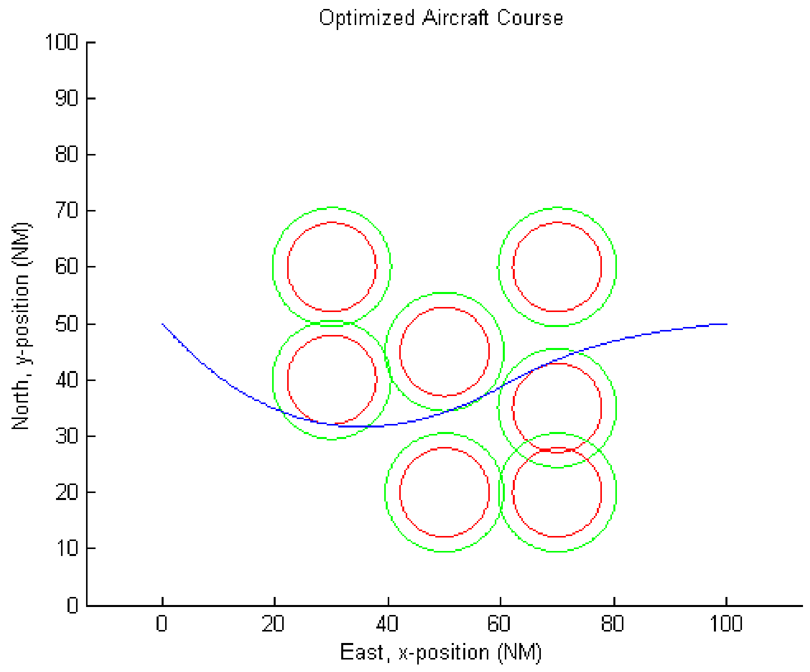

Figure 4.10: TCSV Seven Radar Optimized Course

Table 4.9: TCSV Seven Radar Problem Initial Guess

East (NM)	North (NM)
0	50
30	25
70	45
100	50

Table 4.10: TCSV Seven Radar Problem Final Cost and Run Time

Cost	CPU Run Time (sec)
2.7818	9.40

code performs. To begin, there are a few quirks in how it operates. There is one debilitating glitch that occurs when the user provided initial guess goes through any exclusion zone. The infeasibility of this path creates a plateaued cost function on which it is impossible for GPOPS and SNOPT to find a gradient. It is a simple glitch that is easy to avoid by picking an initial guess that avoids any no-fly zones, a task that would be expected regardless of this glitch's existence.

The costs from the above examples are all very similar. Even the seven radar example cost is only a few tenths higher than the single radar problem. This is due to how the final time dominates the cost. Section 3.2.2 discussed the weights of each cost components, with the final time having a weight 10× greater than the radar cost's weight. As the flight time for these examples hovered around 0.23 hours, the cost hovered around 2.3. The weights here were picked for how they caused the problem to behave for the examples used to test ACOT's feasibility. The examples used here are clearly not realistic scenarios, they exist only as test. The radars are far too weak and the flight distances very short. Because of this, the weights are not meant to be used by an end user. Instead a user should pick what weights create solutions they can use.

4.1.5 Three Component State Vector Feasibility. Based on the results in the examples above it is clear that the three component state vector version of ACOT can successfully optimize a wide range of courses around various radar threats. It does so with the speed and regularity necessary for quick mission planning. Obviously there are limits to the code, but, as there are an infinite number of possible problems, testing everything is impossible. With that, it is feasible to use GPOPS to create an aircraft course optimization tool. However, the success of this three component state vector version brought up the question of how advance of a problem could GPOPS handle. With that the five component state vector version of ACOT was created.

4.2 Five Component State Vector Results

Created out of a desire for more realism in the problem, the FCSV ACOT includes more states, more complex dynamics, modified cost function, and the ability to define the problem better through additional constraints. As with the TCSV examples above, the first example problem for this new version is a simple single radar problem before moving on to more complex problem. Unlike the TCSV results, however, the outcome from these runs is not as guaranteed and is plagued by various errors and only partially optimized solutions. As the complexity of the problem increases, the change of a successful optimization decreases.

4.2.1 FCSV Single Radar Problem. While the real world seldom presents threat areas consisting of just one radar, the simplicity of the situation allows for an easy demonstration of ACOT's core capabilities. This simple scenario is also important is it considers the more complex dynamics, but still keeps the problem easy for the optimizer. The first example here is the same as used for the first TCSV example in Section 4.1.1 save for the new aircraft model seen in Table 4.11. This new aircraft model includes the file that contains the radar cross section of the aircraft as a function of angle. While that RCS could be anything, and in reality would be a complex shape, it is kept a constant 5 m^2 for the following problems, unless otherwise noted.

The FCSV code requires more than just the the initial and final positions to be defined at the start and end of the problem. As said in Section 3.4.1 every state must

Table 4.11: FCSV Aircraft Properties

Type	Units	Value
Minimum Speed	kts	200
Maximum Speed	kts	450
Max Acceleration	kts/s	50
Max Deceleration	kts/s	25
Max Load Factor	g's	3
Max Bank Rate	deg/s	4.5
RCS File	m^2	RCS55.mat

be defined at every guess point, and the initial and final positions are still guess points. However, rather than allowing the "Guess Enhancer" to set every state value at these points the user has control over them. Additionally, the FCSV version is designed so that a user has the ability to constrain the heading and velocities at the start and end points in the event they needed to match some specific boundary conditions. If they heading and velocity are not constrained their assigned values do not matter as the optimizer will attempt to make them whatever value is optimum. The positions are automatically constrained since otherwise the problem would not be fully defined. For this first single radar example the initial and final states are in Table 4.12.

Type	Units	Value	Constrained
x_0	NM	0	Yes
y_0	NM	50	Yes
v_0	kts	400	No
h_0	degrees	40	No
x_f	NM	100	Yes
y_f	NM	50	Yes
v_f	kts	400	No
h_f	degrees	90	No

Table 4.12: FCSV Single Radar Problem Initial Position, Guess Points, and Final Position

In this example everything except the number of states and the aircraft model match the first TCSV example. Running ACOT with these values creates the guess plot in Figure 4.11 and the optimized results in Figure 4.12. Figures 4.13 and 4.14 display the aircraft heading and velocity as a function of time. Note that the initial and final values do not match those shown in Table 4.12. As the heading and velocity were not constrained for this problem GPOPS was able to modify them to whichever values produced the lowest cost solution.

As with the TCSV examples, the solid blue line in the course figures represents the aircraft course, the red inner circle the exclusion zone around a radar , and the green outer circle denotes the range around a radar in which the aircraft is said to have been detected.

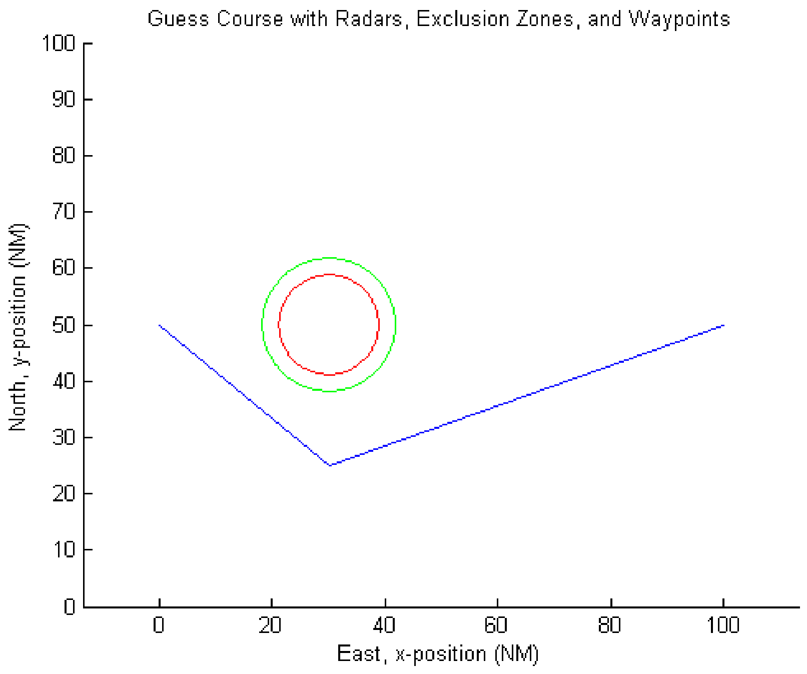

Figure 4.11: FCSV Single Radar Problem Guess Course

The cost and run time for this problem are shown in Table 4.13. Not the slightly higher cost than with the TCSV as the code does consider more variable when calculating the total cost. Additionally, the run time is dramatically increase. This trend continues as the complexity increase.

Table 4.13: FCSV Single Radar Problem Final Cost and Run Time

Cost	CPU Run Time (sec)
2.3833	34.1615

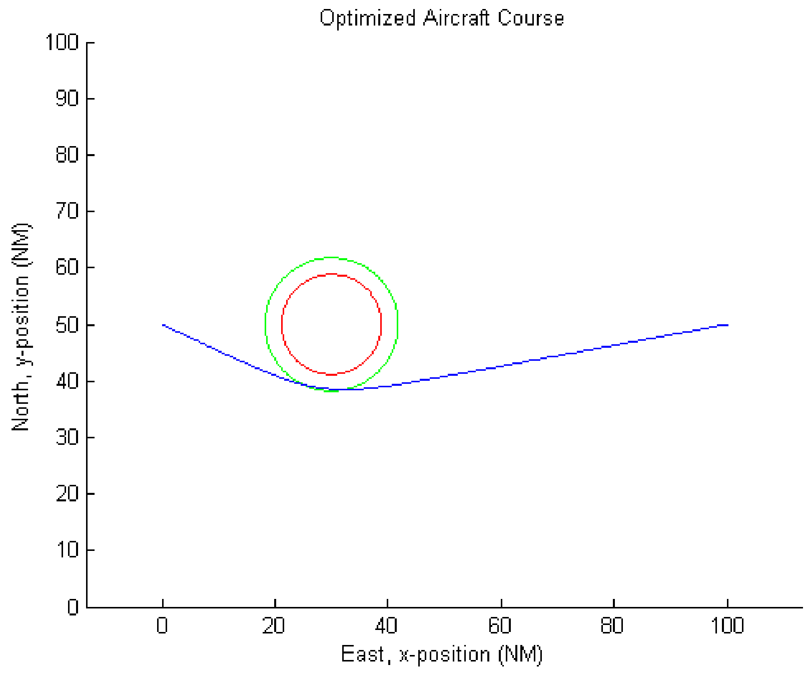

Figure 4.12: FCSV Single Radar Problem Optimized Course

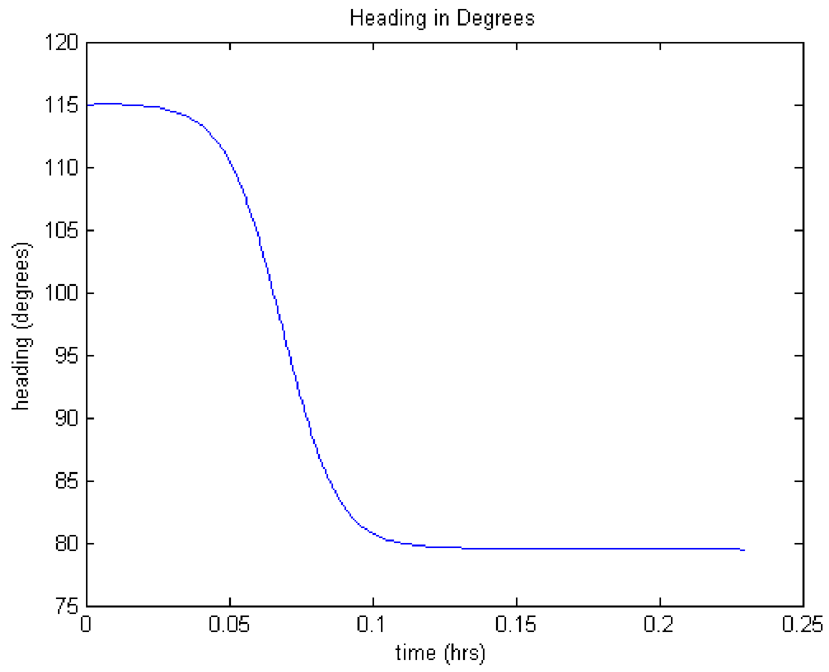

Figure 4.13: FCSV Single Radar Problem Aircraft Heading

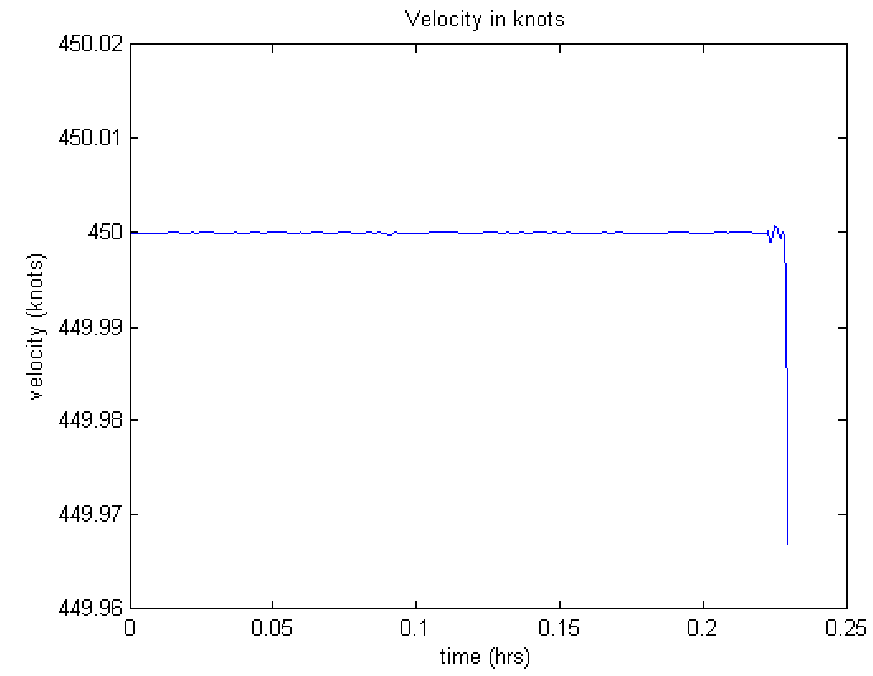

Figure 4.14: FCSV Single Radar Problem Aircraft Velocity

Even though the FCSV and TCSV problems use different dynamics and different cost functions, it is interesting to see how different the optimized courses between the two versions of code are. As seen in Figure 4.14, the aircraft remained at its maximum speed of $450kts$, which matches the constant velocity of the TCSV example. Note that the aircraft RCS and the initial guess for the two versions are the same; therefore the differences between the two courses result more from the dynamics and cost function rather than the aircraft or initial guess. Figure 4.15 shows the output from both the three and five component state vector runs, with the solid blue line as the result from the three state run and the dashed red the result from the five state run. It appears then the slight differences in the aircraft model and its maximum control values, as well as the variations in the cost function, force the aircraft to travel slightly closer to the radar, but overall the optimized course is the same. This demonstrates the changes to the dynamics and the aircraft model do not drastically modify the optimized solution.

Just as with the TCSV example, both the dynamics equations and the control vectors of this FCSV run were used in MATLAB's® ODE45 solver. In the case of this

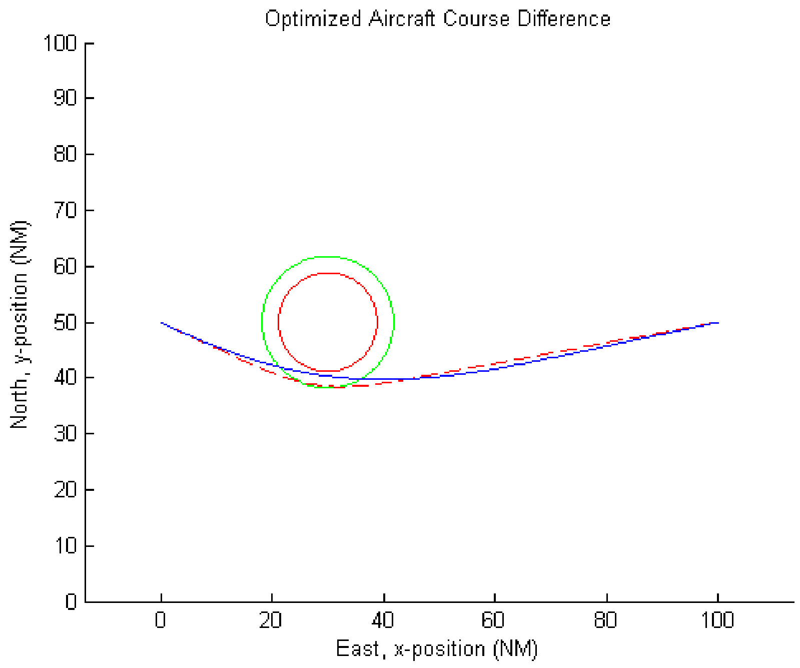

Figure 4.15: FCSV Single Radar Problem Optimized Difference (TCSV: Blue Solid, FCSV: Red

single radar problem the ODE course matched the GPOPS course well, but not as closely as the same test for the TCSV problem. Figure 4.16 shows a slight deviation around the main bend in the course. This deviation most likely occurred not due to bad dynamics, or even a bad control vector, but because of how the control vector is interpolated during ODE45's run. As the control vector is not defined for every point in time, its value at an arbitrary time must be interpolated based on the value at the times closest to ODE45's current point of interest. In the end, the successful convergence of ODE45 shows that, as far as the dynamics used in the FCSV version of ACOT are concerned, the output is a feasible, flyable course.

4.2.1.1 Single Radar Problem with Alternate Initial Guess. A quick demonstration of how the the final outcome is still dependent on the initial guess, the example from Section 4.1.2 is repeated here using the FCSV dynamics with the results shown in Figures 4.17 and 4.18. While it does not produce the exact result

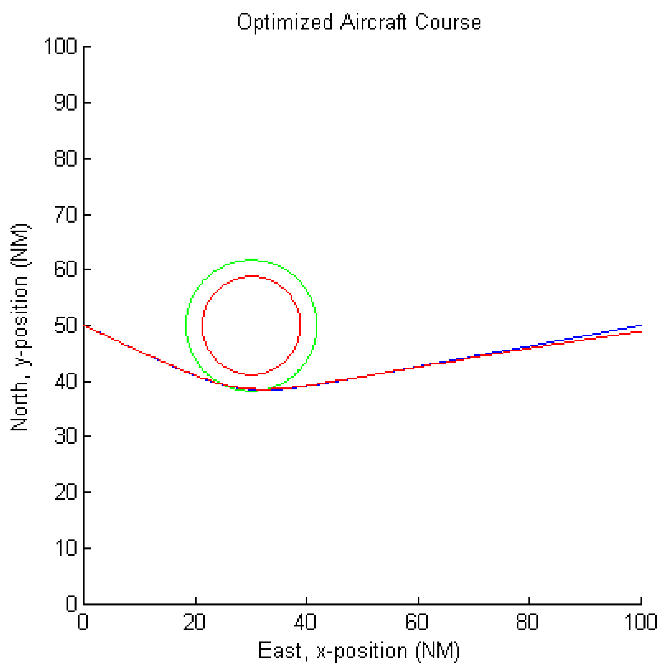

Figure 4.16: FCSV Single Radar Problem ODE Comparison

as the TCSV example it does confirm the effect the initial guess has on the optimum solution acts there same for the more complex dynamics here.

4.2.2 FCSV Three Radar Problem. The next step up from the single radar problem is the three radar scenario. To keep the number of different problems down, this too is a copy of a TCSV example problem, in this case from Section 4.1.3. The differences between the two problems lie only in the size of the state vector and the cost function. The initial and final positions, as well as the positions of the intermediate points are the same. The results, shown in Figures 4.19 and 4.20

This example ran in 72.15 seconds, just over twice as long as the single radar problem, and resulted in a completely optimized output. Combine that with its small cost of 2.3822 and this example appears successful, which it is. The three radars in this example allow for verification of the cost function in the same manner it was performed for the TCSV in Section 4.1.3. However, difficulties in the optimization process prevented two perturbed courses from being calculated, and the one that

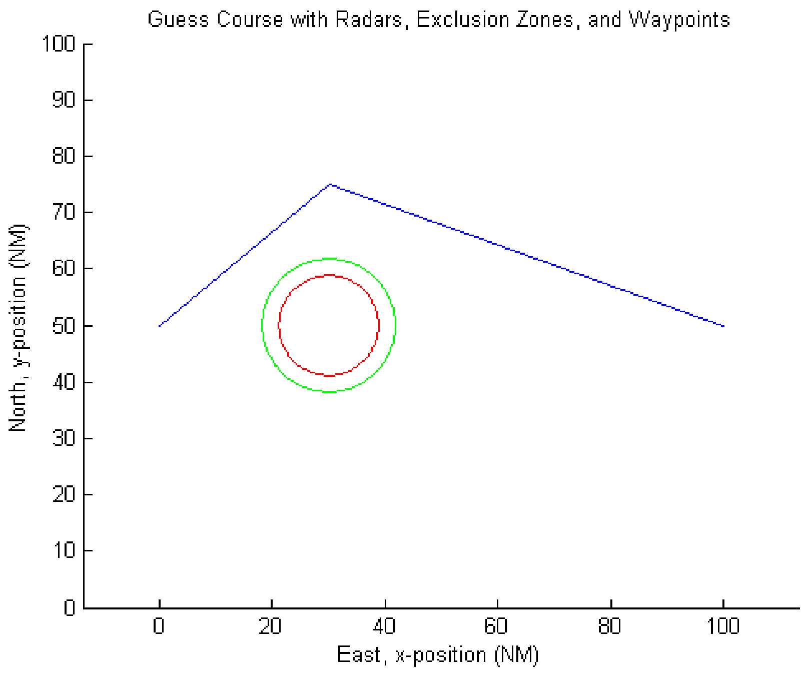

Figure 4.17: FCSV Single Radar Problem with Alternate Initial Guess

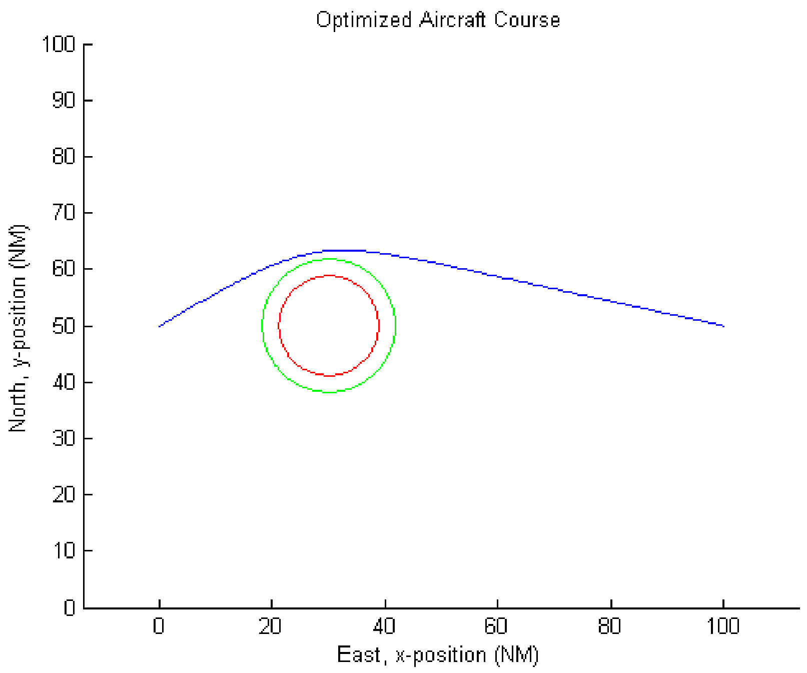

Figure 4.18: FCSV Single Radar Problem with Alternate Initial Guess Optimized Course

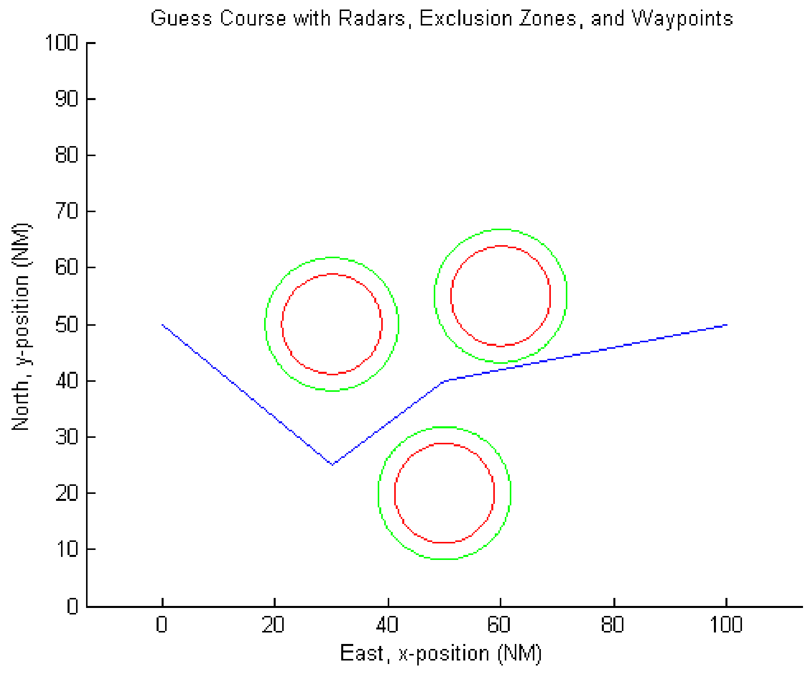

Figure 4.19: FCSV Three Radar Problem Guess Course

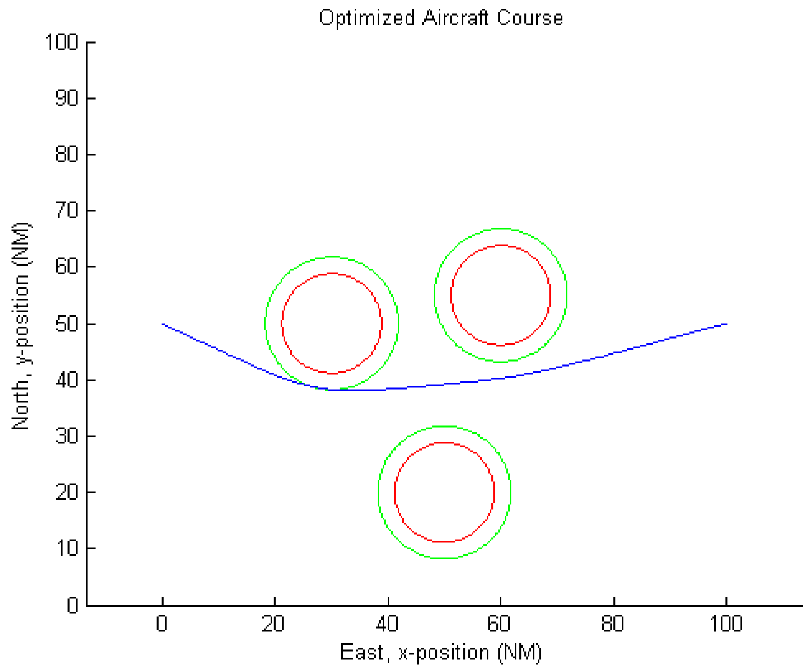

Figure 4.20: FCSV Three Radar Problem Optimized Course

was calculated is not fully optimized. The difficulties that caused this failure in optimization will be discussed in Section 4.2.3, where it will become clear that the success of this and the previous example is, when it comes to FCSV, unusual. For the cost verification, Figure 4.21 shows the two courses, with the solid blue as the true course and the dashed red as the perturbed, unoptimized course, while Table 4.14 shows the two costs. This is an extreme example of how different courses will have different cost function, but the cost function is shown to be valid.

Table 4.14: FCSV Cost Function Verification

GPOPS Output	2.3848
Perturbed	8.6489

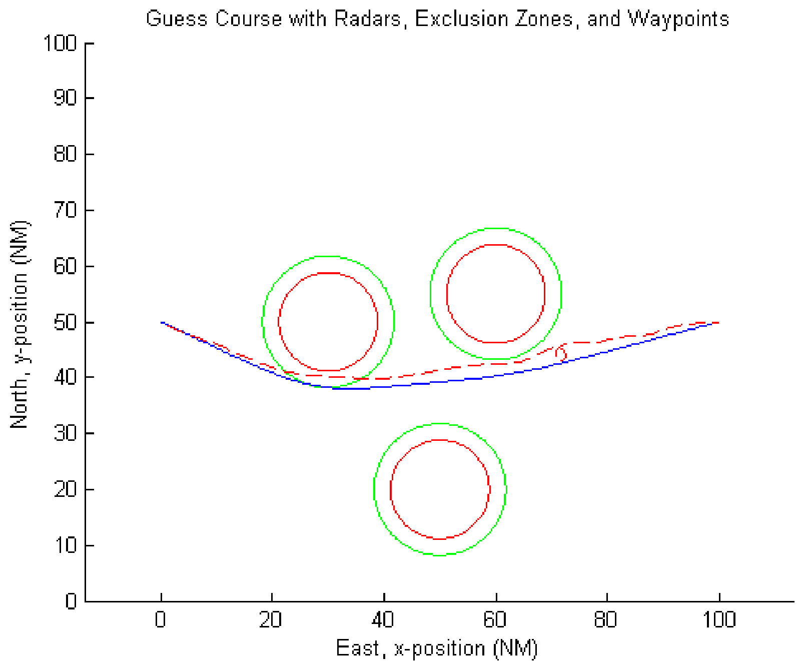

Figure 4.21: FCSV Cost Verification Courses

4.2.3 Failed FCSV Examples.

Following the successful optimization of the single radar problem in Section 4.2.1 it was necessary to test the various constraints the FCSV version of ACOT could place on the problem. The first test required constraining the initial and final headings. This, while at first glance a simple change, caused the code to fail in its optimization. More specifically, its run time far surpassed the two successful optimization above before SNOPT, the internal optimizer inside GPOPS, stopped its optimization and output a message claiming it experienced numerical difficulties. The exact nature of these numerical difficulties is unknown, but they most likely resulted from the large difference between the constrained headings and the optimum headings. SNOPT attempted to follow the gradients it was calculating, but GPOPS kept sending back heading values for the start and end points that caused discontinuities inside SNOPT. Because SNOPT could not rectify the issue it claimed numerical difficulties and gave up. Close examination of the heading results from this run in figure 4.22 show that the output headings approached, but did not reach, the constrained values of 45° for the start and 135° at for the final position.

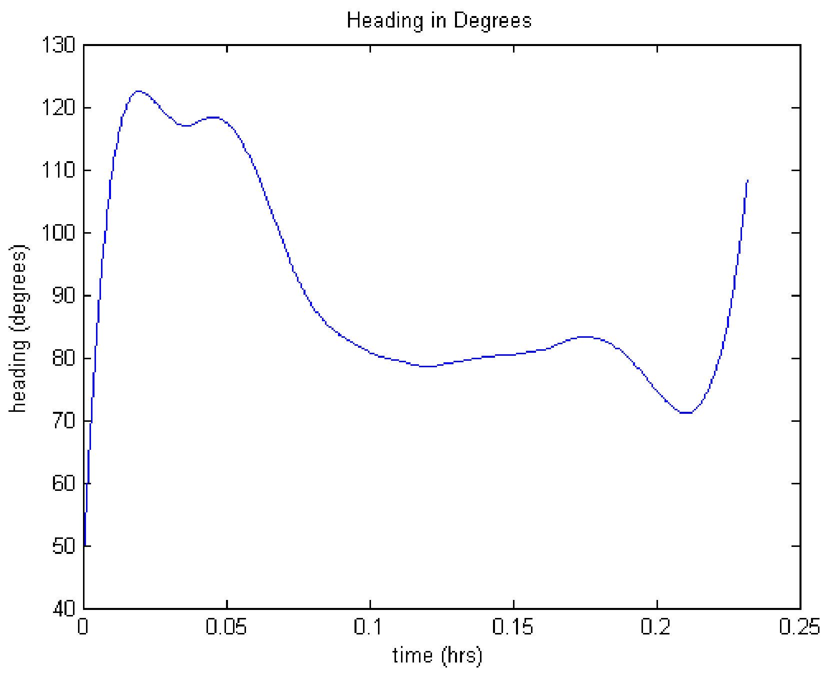

Figure 4.22: FCSV Constrained Heading Results

A similar problem occurred when the initial and final velocities were constrained to a value less than the aircraft maximum speed of 450kts. The code ran, but rather than outputting a fully optimized solution SNOPT stopped part way through and claimed numerical difficulties. Once again the best explanation for this is the difference between the constrained and optimum values. Figure 4.23 shows that the aircraft came close to starting at its assigned speed of $350kts$ and approached the same speed at its target destination, but examination of the actual MATLAB® reveals they were not exact. At the very least this does demonstrate that the code can handle accelerations in the problem, but that in the end the variability proves too much for the optimizer.

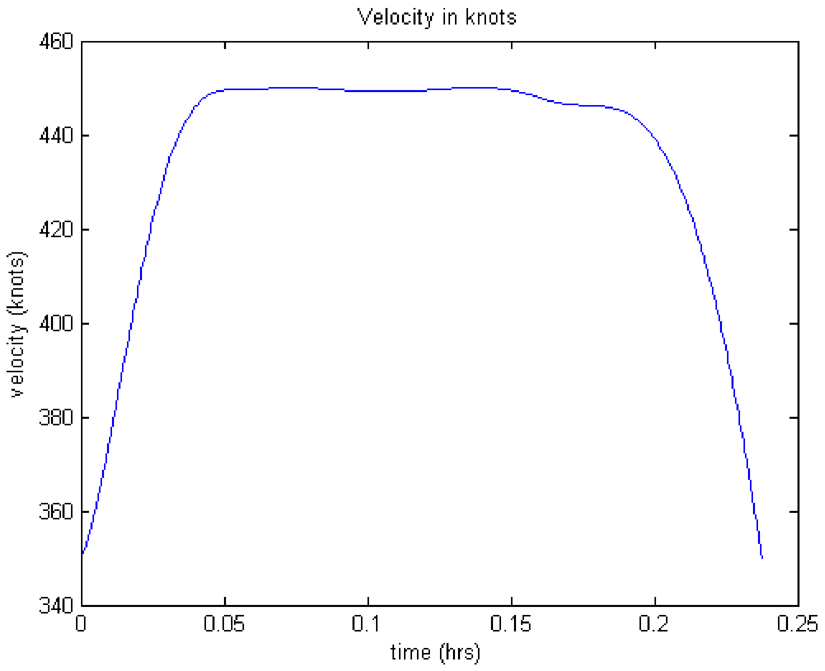

Figure 4.23: FCSV Constrained Velocity Results

The numerical difficulties also extends to the desired final time problem. Where all the examples so far finished in roughly 0.23 hours this examples was set to end around 0.3 hours. While a human mission planner could first consider slowing down, Figure 4.24 shows that rather than slowing down the aircraft did a full 360° turn near the end of the problem. This results hints at the underlying problem with really all of these failed examples: too many possible solutions. When it comes to the desired final time problem, or really any of the failed examples, there are several different ways that the optimizer could approach the problem. For the desired final time problem specifically the course could have flow farther out from the radar, the aircraft could have slow down, or it could do something like the loop that it did do. Furthermore, the coupled nature of the dynamics, where, for example, a slower speed will increase the heading change rate for a fixed bank angle, allows for too much to change with one slight adjustment. There is too much coupling.

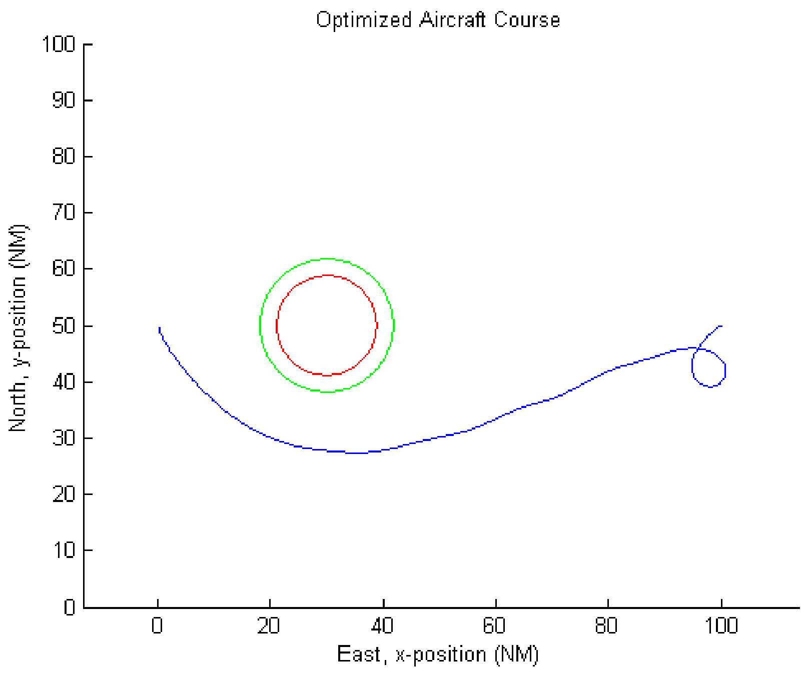

Figure 4.24: FCSV Desired Final Time Results

This coupling issue is revealed even more when a non-circular radar cross section is added to the problem. Figure 4.25 shows a two lobe radar cross section with values of 1 m^2 at the aircraft nose and tail, and $6m^2$ at the wingtips. When the optimizer adjusts the aircraft's bank rate, that modifies the aircraft bank angle, which then plays into the heading change. That heading change directly affects the aircraft heading, which is used to calculate which angle on the aircraft is facing the current radar of interest. That angle is used to look up the RCS value in the provided RCS value table that is then used in the calculation of the radar cost. That is for just one collocation point and one radar. When this problem is expanded to dozens of collocation points and multiple radars it is a daunting problem, one that is too much for GPOPS and SNOPT.

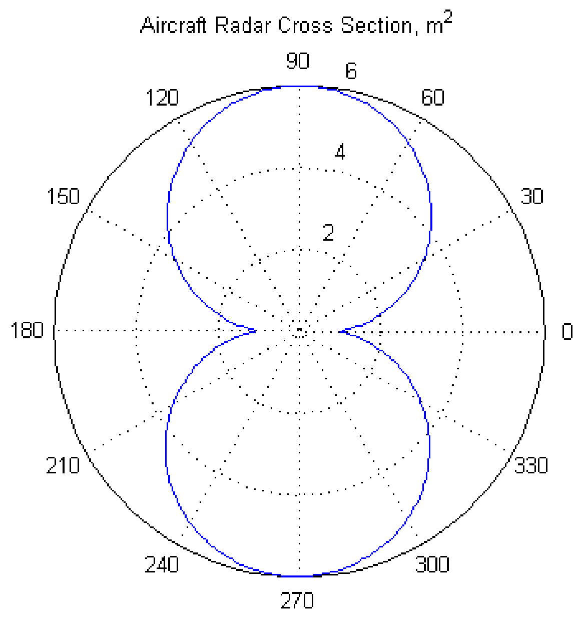

Figure 4.25: FCSV Two Lobe Radar Cross Section

An additional possible cause for the trouble the code experiences with the two lobe radar cross section is the discontinuity where the RCS is $1m^2$, however it is thought this is ignored due to the discrete manner in which GPOPS checks the RCS. Even if it pulled the cross section values for exactly $0°$, $1°$, and $359°$ all it would see are three separate cross sections sizes at those points. With those stored it would attempt to incorporate them into the cost gradient. It is this incorporation that is the most likely cause of the difficulties.

4.2.4 Five Component State Vector Feasibility. These examples are just a few of the dozens of problems run during the development of the FCSV ACOT. Some, like the two successful version in Section 4.2.1 and 4.2.2, worked. The majority however experienced numerical difficulties and stopped like those in this section, or they never finished running. One particular case ran over a 48 hour period and never stopped, as SNOPT had gotten into an internal minor iteration loop. Other cases caused GPOPS to add more and more collocation points until the computer memory filled and MATLAB® canceled the run. In the end, the added realism of the FCSV subtracted from the overall success of the code. As the dynamics improved, the run time increased and the success rate fell. As the cost function and dynamics grew to reflect the real world, SNOPT began to struggle. While it is possible to get the simplest of problems to run with the FCSV, in the end it proves infeasible. Any aircraft course optimization must rely on the TCSV.

V. Conclusions

This research's ultimate goal was to determine the feasibility of using GPOPS as the core component of an aircraft course optimization tool. Based on the results in Chapter IV it can be reasonably determined that using GPOPS for this type of problem is possible, though only when using simplified dynamics. The addition of any noteworthy amounts of realism, be it in the aircraft model or the flight dynamics, and the complexity of the problem causes GPOPS and SNOPT to stall.

The determination of this problem's feasibility rested on the creation of a tool that, with GPOPS at its core, would take a rough course for an aircraft and optimize it around various threat areas. Early work towards that goal created versions of the code that worked successfully, and in fact it was by retreating to those versions that this research could be considered a success. Those early versions lead to more advanced code that incorporated more realistic aircraft models and flight dynamics. However, in the end the advance code failed.

Section 4.1 demonstrated how the simple three component state vector accomplished this thesis' goal. Section 4.2 shows that the advanced dynamics in the five component state vector could work under certain situation, but ultimately failed when tasked with anything truly useful.

The next step for ACOT is the determination of the best weights to apply on each component of the cost function. Mentioned before, the weights used for the testing and examples were picked for how well they worked for the small radars and limited ranges involved in these test problems. Real world scenarios will require different weights.

The successfully creation of a GPOPS based aircraft course optimizer should not be the end of this line of research. Further work should focus on creating an optimizer that can make use of more advanced aircraft models and dynamics. The simple moving point of ACOT is barely enough to begin mission planning, and as the types of threats change, and the number of them grow, it will take much more than GPOPS and SNOPT to find the best way around them. The advanced dynamics in the FCSV caused GPOPS and SNOPT to struggle because of the extremely coupled natured of the equations. Trying to determine the relationship between all the components, and then minimizing the cost each added to the total, is too much for the general nature of SNOPT. Further work on this should involve the creation of an optimizer uniquely suited to the complex world of aircraft flight.

Crafting a tool to achieve the goal of improving mission planning times via the ability to take a rough idea for an aircraft course and quickly optimize is an area well worth further research. This experiment has shown GPOPS can handle a simple version of the task, but that ultimately any work with GPOPS is limited by its general nature. GPOPS may be feasible, and ACOT usable, but only if the user is willing to accept the limited parameters it considers.

Appendix A. Three State Matlab Scripts

A.1 Three State Shell Code

Listing A.1: Appendix2/courseOptShell.m

```
% -------------------------
% Course Optimization Problem Shell Script
% -------------------------
%
% Aircraft Course Optimization Tool (ACOT)
%
% A script to aid interfacing the aircraft course optimization ...
   GPOPS
% problem designed by 2Lt Ryan Gauntt with an GUI interface. As a ...
   stand
% alone tool, this shell can also act as a method for running ...
   problems with
% the ACOT
%
% Globals:
%   INPUTS - structure containing all the necessary variables and ...
   conditions
%   for ACOT to run.
%   FLAGS - structure containg all the flags that will allow logic...
    later on
%   in the code to run.
% Variables
%   Ground_Speed - ground speed of aircraft (nm).
%       For now ACOT uses a constant
%       speed and turn radius model. Later versions may use speed ...
   as a
%       control or state variable.
%   Turn_Radius - turn radius of aircraft at set Ground_Speed (nm)...
   .
%       Granted, a turn radius is based on airspeed and several ...
   other,
%       factors but this number is just a base one for ACOT to use...
   . As ACOT
%       matures this variable may become a function based on speed...
   .
%   RCS - name of file containing radar cross section of aircraft.
%       File must currently be a .mat file with the first column ...
   angles in
%       radians and the second column the RCS of the aircraft at ...
   that angle
%       (m^2)
%   radar - a matrix containing all the information about each ...
   radar. This
%   will be used with the radar equation later on to produce the ...
   maximum
%   radar range. Additionally, it can be used as part of the cost ...
   function
```

```
%   as well. NOTE ON USAGE: each row corresponds to a radar. To ...
    add more
%   radars type more rows in the matrix. Variable used in matrix ...
    are
%   discussed below:
%       x - radar x position in local cooridnate frame (nm)
%       y - radar y position in local cooridnate frame (nm)
%       Ps - Power (average) transmitted from source (watts)
%       Gs - Gain of the source antenna (dB)
%       Gm - Gain of the receiving antenna (dB)
%       f - frequency of radar beam (kHz)
%       kTBn - measure of the internal noise power of the receiver...
     (dB)
%       L - loss factor (dB)
%       SNR - Signal to Noise ratio limit for detection (dB)
%       KillSNR - SNR that signifies you will now die from ...
    something (dB)
%   Initial Condition
%       x0 - aircraft's initial x position (nm)
%       y0 - aircraft's initial y position (nm)
%   Final Condition
%       xf - aircraft's final x position (nm)
%       yf - aircraft's final y position (nm)
%   GuessPoints - The x,y coordinates that provide a reasonable ...
    guess for
%   the path the optimizer will eventually find. Recommended this ...
    be a
%   voronoi path with three points for each radar crossing (before...
     hiting
%   radars, in the middle of the two radars, and after leaving the...
     radars).
%   These are NOT waypoints that the plane has to hit, save for ...
    the initial
%   and final positions, but those are deemed necessary not by the...
     guess
%   part of the code.
%   Waypoints - Contains the row number of any guessPoint that is
%   considered a mandatory waypoint, followed by the allowable ...
    circular
%   error around that waypoint.
% -------------------------

close all
clear all
clc

global INPUTS FLAGS

% Aircraft Information
Ground_Speed = 450;
Turn_Radius  =   2;
RCS = 5;
```

```
% Radar Information
%        [x    y   Ps Gs Gm f kTBn L SNR KillSNR]
radars = [30 50 40000 40 40 9.9931e+006 -120 3 13 18;
          50 20 40000 40 40 9.9931e+006 -120 3 13 18;
          60 55 40000 40 40 9.9931e+006 -120 3 13 18];
          %25 25 40000 40 40 9.9931e+006 -120 3 13 18];
          %70 90 40000 40 40 9.9931e+006 -120 3 13 18;
          %70 50 40000 40 40 9.9931e+006 -120 3 13 18;
          %30 90 40000 40 40 9.9931e+006 -120 3 13 18];

radar_constant = 33; %Will be a subtraction

% Initial Conditions
x0     = 0;
y0     = 50;

% Final Conditions
xf     = 100;
yf     = 50;

% GuessPoints
            %[x1 y1; x2 y2;...] First and last guessPoints will ...
               always need
            %to be x0 y0 and xf yf
guessPoints = [x0 y0;
               30 25;
               50 40;
               xf yf];

% Plots (1 = yes, 0 = no)
Plot_Guess       = 1;
Plot_Radar_SNR   = 0;
Plot_Solution    = 1;
Plot_SolandSNR   = 0;
Plot_Colocation  = 0;
Plot_Heading     = 1;
Plot_hdot        = 0;

C_AN = [0 1 0;
        1 0 0;
        0 0 -1];
INPUTS.C_NA = C_AN^-1;

% Convert Radar input into usable numbers (freq (kHz) to ...
   wavelength (nm))
for i = 1:size(radars)
    radars(i,3) = 10*log10(radars(i,3));
    radars(i,6) = 10*log10((299710000./(radars(i,6).*1000))^2); %...
       converts from kHz frequency to m wavelenght
end
```

```
%Put all above variable into necessary structures
maxRCS = RCS;
INPUTS.aircraft.velocity = Ground_Speed;
INPUTS.aircraft.turnRadius = Turn_Radius;
INPUTS.RCS = RCS;
for i = 1:size(radars)
    %Due to the prevelance of metric units in EM math, all range ...
       and SNR
    %calculations are done using the metric system. Even so, all ...
       units and
    %computer logic use nautical miles.
    INPUTS.radarMaxRange(i,:) = (10.^((radars(i,3)+radars(i,4)+...
       radars(i,5)+radars(i,6)+...
        maxRCS-radar_constant-radars(i,7)-radars(i,8)-radars(i,9))...
           ./10)).^(1/4)*5.39956803*10^-4;
    if radars(i,10) == 0
        INPUTS.radarKillRange(i,:) = 0;
    else
        INPUTS.radarKillRange(i,:) = (10.^((radars(i,3)+radars(i...
           ,4)+radars(i,5)+radars(i,6)+...
            maxRCS-radar_constant-radars(i,7)-radars(i,8)-(radars(...
               i,10)))./10)).^(1/4)*5.39956803*10^-4;
    end
end
INPUTS.radarInfo = radars;
INPUTS.radarConstant = radar_constant;
INPUTS.state.x0 = x0;
INPUTS.state.y0 = y0;
INPUTS.state.xf = xf;
INPUTS.state.yf = yf;
[nuTime, nuGuess, nuControl] = courseOptGuessEnhancer(INPUTS, ...
   guessPoints);
INPUTS.guess.time = nuTime;
INPUTS.guess.state(:,1:3) = nuGuess;
INPUTS.guess.control = nuControl;

plotMin = min([x0,xf,y0,yf,radars(:,1)',radars(:,2)']);
plotMax = max([x0,xf,y0,yf,radars(:,1)',radars(:,2)']);

INPUTS.plotLim.min = plotMin;
INPUTS.plotLim.max = plotMax;

if Plot_Radar_SNR == 1 || Plot_SolandSNR == 1
    disp('Calculating meshgrid values for Radar SNR Plot')
    disp('If this plot is not desired, set Plot_Radar_Cost and/or ...
       Plot_SolandRad equal to 0');
    prog = 0;
    h = waitbar(prog,sprintf('%3.0f%% Complete',prog));
    [x y] = meshgrid(plotMin:0.1:plotMax, plotMin:0.1:plotMax);
    radStats = radars(1:8);
    cost = [];
    progmax = size(radars,1)*3+1;
```

```
for i = 1:size(radars,1)
    for j = 1:size(x,1)
        for k = 1:size(y,2)
            Rsqd(j,k) = (x(j,k)-radars(i,1))^2+(y(j,k)-radars(...
                i,2))^2;
        end
    end
    prog = prog + 1/progmax;
    waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
    [j k] = size(Rsqd);
    S = ones(j,k);
    CS = max(RCS(:,2));
    Q = (radars(i,3)+radars(i,4)+CS+radars(i,5)+radars(i,6)...
        -...
    radar_constant-radars(i,7)-radars(i,8)).*S;
    for j = 1:size(x,1)
        for k = 1:size(y,2)
            Rdb(j,k) = 10.*log10((Rsqd(j,k)*1852^2).^2);
        end
    end
    prog = prog + 1/progmax;
    waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
    SNR = Q-Rdb;
    if isempty(cost)
        cost = SNR;
    else
        cost = cost+SNR;
    end
    prog = prog + 1/progmax;
    waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
end
for j = 1:size(cost,1)
    for k = 1:size(cost,2)
       if cost(j,k) > min(radars(:,10));
           cost(j,k) = min(radars(:,10));
       end
    end
end
prog = prog + 1/progmax;
waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
close(h)
if Plot_Radar_SNR == 1
    figure()
    axis([plotMin plotMax plotMin plotMax])
    G = surf(x,y,cost,'FaceColor','interp','FaceLighting','...
       phong');
    set(G, 'linestyle', 'none');
    title('Radar Cost Function Field')
    axis equal
    xlabel('East, x-position (NM)')
    ylabel('North, y-position (NM)')
    zlabel('Signal to Noise Ratio (dB)')
```

```
            view(2)
        end
        INPUTS.x = x;
        INPUTS.y = y;
        INPUTS.radPlots.cost = cost;

    end

    if Plot_Guess == 1
        figure()
        hold on
        axis([plotMin plotMax plotMin plotMax])
        for i = 1:size(radars)
            circle(radars(i,1:2),INPUTS.radarKillRange(i,:),250,'r');
            circle(radars(i,1:2),INPUTS.radarMaxRange(i),250,'g');
        end
        for j = 1:size(INPUTS.guess.state(:,1:2),1)-1
            plot([INPUTS.guess.state(j,1),INPUTS.guess.state(j+1,1)],[...
                INPUTS.guess.state(j,2),INPUTS.guess.state(j+1,2)]);
        end
        axis equal
        title('Guess Course with Radars, Exclusion Zones, and ...
            Waypoints')
        xlabel('East, x-position (NM)')
        ylabel('North, y-position (NM)')
    end

    pause(0.5)

    disp('Plots are behind the MATLAB window')
    reply = input('Do you want continue with current guess and radar ...
        field? Y/N [Y]: ', 's');
    if ~strcmp(reply,'Y') && ~strcmp(reply,'y')
        error('Run Cancelled')
    end

    FLAGS.solutionPlot = Plot_Solution;
    FLAGS.solutionAndRadarSNR = Plot_SolandSNR;
    FLAGS.colocationPlot = Plot_Colocation;
    FLAGS.hdotPlot = Plot_hdot;
    FLAGS.headingPlot = Plot_Heading;


    courseOptMain
```

A.2 Three State Guess Enhancer Code

Listing A.2: Appendix2/courseOptGuessEnhancer.m

```
function [nuTime, nuGuess, nuControl] = courseOptGuessEnhancer(...
    INPUTS, guessPoints)
```

```
%-------------------------------------
% BEGIN: function courseOptGuessEnhancer.m
%
% This script takes the main waypoints provided in the ...
   courseOptShell
% script and added in additional points between them to provide a ...
   more
% accurate initial guess to the GPOPS algorthim.
%
%-------------------------------------

N = 1; % Number of steps to add between each waypoint

providedVelocity = INPUTS.aircraft.velocity;

K = size(guessPoints,1)-1;
nuGuess = [guessPoints(1,1:2),atan2(guessPoints(2,2)-guessPoints...
   (1,2),guessPoints(2,1)-guessPoints(1,1))];
for i = 1:K
    dist(i) = norm(guessPoints(i+1,1:2)-guessPoints(i,1:2));
    deltaTime(i) = dist(i)/providedVelocity;
    h = atan2(guessPoints(i+1,2)-guessPoints(i,2),guessPoints(i...
       +1,1)-guessPoints(i,1));
    deltaDist = dist(i)/N;
    for j = 1:N
        nuGuess = [nuGuess; nuGuess(end,1)+cos(h)*deltaDist,...
           nuGuess(end,2)+sin(h)*deltaDist,h];
    end
end
totalTime = sum(deltaTime(1,:));
delTime = totalTime/(N*K);

nuTime = 0;
nuControl = 0;
for i = 1:N*K
    nuTime(i+1) = nuTime(end)+delTime;
    nuControl(i+1) = 0;
end

nuTime = nuTime';
nuControl = nuControl';
```

A.3 Three State GPOPS Main Code

Listing A.3: Appendix2/courseOptMain.m

```
% -----------------------
% Course Optimization Problem
% -----------------------

% The problem solved here is given as follows:
% Minimize
%       t_f, J
% subject to the dynamic constraints
%       dx/dt = v*cos(u)
%       dy/dt = v*sin(u)
%       dth/dt = u
% with the boundary conditions
%       x(0) = 0, y(0) = 0, v(0) = 450
%       x(t_f) = 25, y(t_f) = 50, v(t_f) = 450
%
%  --------------------------------------------------

tic

global CONST INPUTS FLAGS

CONST.radarPos = INPUTS.radarInfo(:,1:2);
CONST.radarStats = INPUTS.radarInfo(:,3:9);
CONST.radarMaxRange = INPUTS.radarMaxRange;
CONST.radarKillRange = INPUTS.radarKillRange;
CONST.radarConstant = INPUTS.radarConstant;
CONST.v = INPUTS.aircraft.velocity;
CONST.R = INPUTS.aircraft.turnRadius;
CONST.RCS = INPUTS.RCS;

x0   = INPUTS.state.x0;
y0   = INPUTS.state.y0;
xf   = INPUTS.state.xf;
yf   = INPUTS.state.yf;
xmin   = min(INPUTS.guess.state(:,1));
xmax   = max(INPUTS.guess.state(:,1));
ymin   = min(INPUTS.guess.state(:,2));
ymax   = max(INPUTS.guess.state(:,2));
hmin = -2*pi;
hmax = 2*pi;
param_min = [];
param_max = [];
path_min   = CONST.radarKillRange.^2;
path_max   = xmax^2*ones(1,size(CONST.radarKillRange))';
event_min = [];
event_max = [];
duration_min = [];
duration_max = [];


limits.meshPoints = [-1 +1];
limits.nodesPerInterval = 10;
limits.time.min = [0 0];
limits.time.max = [0 10];
limits.control.min     = -CONST.v/CONST.R;
limits.control.max     = CONST.v/CONST.R;
limits.state.min(1,:) = [x0 xmin-100 xf];
limits.state.max(1,:) = [x0 xmax+100 xf];
limits.state.min(2,:) = [y0 ymin-100 yf];
limits.state.max(2,:) = [y0 ymax+100 yf];
limits.state.min(3,:) = [hmin hmin hmin];
limits.state.max(3,:) = [hmax hmax hmax];
limits.parameter.min  = param_min;
limits.parameter.max  = param_max;
limits.path.min       = path_min;
limits.path.max       = path_max;
limits.event.min      = event_min;
limits.event.max      = event_max;
limits.duration.min   = duration_min;
limits.duration.max   = duration_max;

guess.time(:,1)       = INPUTS.guess.time;
guess.state(:,1)      = INPUTS.guess.state(:,1);
guess.state(:,2)      = INPUTS.guess.state(:,2);
guess.state(:,3)      = INPUTS.guess.state(:,3);
guess.control(:,1)    = INPUTS.guess.control;
guess.parameter       = [];

setup.name  = 'CourseOpt-Problem';
setup.funcs.cost = 'courseOptCost';
setup.funcs.dae = 'courseOptDae';
setup.limits = limits;
setup.guess = guess;
setup.derivatives = 'finite-difference';
setup.direction = 'increasing';
setup.autoscale = 'off';
setup.mesh.tolerance = 1e-3;
setup.mesh.iteration = 10;
setup.mesh.nodesPerInterval.min = 4;
setup.mesh.nodesPerInterval.max = 12;

[output,gpopsHistory] = gpops(setup);
solution = output.solution;
solutionPlot = output.solutionPlot;

if gpopsHistory(1,end).output.SNOPT_info == 1
    disp('Optimality conditions satisfied.')
    disp('Acceptable Course')
    disp(['Total Cost of Course: ',sprintf('%3.4f',gpopsHistory...
        (1,1).output.cost)])
else if gpopsHistory(1,end).output.SNOPT_info == 3
        disp('Optimization sucessful, but best accuracy not ...
            acheived.')
        disp('Course is usable, but optimizer had difficulty.')
        disp(['Total Cost of Course: ',sprintf('%3.4f',...
            gpopsHistory(1,1).output.cost)]);
    else if gpopsHistory(1,end).output.SNOPT_info == 41
            disp('Optimization experienced numerical difficulties....
                ')
            disp('Current point cannot be improved, though ...
                solution has been provided.')
            disp('Use of solution possible, though more analysis ...
                is required.')
            disp(['Total Cost of Course: ',sprintf('%3.4f',...
                gpopsHistory(1,1).output.cost)]);
        else
            disp('Unsucessfull Optimization.')
            disp('Ignore Output')
        end
    end
end

toc

for i = 1:size(solutionPlot.state(:,3),1)
        V_A = [cos(solutionPlot.state(i,3)); sin(solutionPlot....
            state(i,3)); 0];
        V_N = INPUTS.C_NA*V_A;
        solutionPlot.state(i,3) = acos(V_N(1,1))*180/pi;
end

if FLAGS.solutionPlot == 1 || FLAGS.solutionAndRadarSNR == 1
    figure()
    hold on
    axis([INPUTS.plotLim.min INPUTS.plotLim.max INPUTS.plotLim.min...
         INPUTS.plotLim.max])
    for i = 1:size(CONST.radarPos)
        circle(CONST.radarPos(i,:),CONST.radarKillRange(i),250,'r'...
            );
        if FLAGS.solutionAndRadarSNR == 1
            circle(CONST.radarPos(i,:),CONST.radarMaxRange(i),250,...
                'k');
        else
            circle(CONST.radarPos(i,:),CONST.radarMaxRange(i),250,...
                'g');
        end
    end
    if FLAGS.solutionAndRadarSNR == 1
        G = surf(INPUTS.x,INPUTS.y,INPUTS.radPlots.cost,'FaceColor...
            ','interp','FaceLighting','phong');
        set(G, 'linestyle', 'none');
        plot3(solutionPlot.state(:,1),solutionPlot.state(:,2),25*...
            ones(size(solutionPlot.state(:,1),1)),'k');
        view(2)
    else
        plot(solutionPlot.state(:,1),solutionPlot.state(:,2));
    end
    if FLAGS.colocationPlot == 1
    plot(solution.state(:,1),solution.state(:,2),'ro');
    end
    axis equal
    title('Optimized Aircraft Course')
    xlabel('East, x-position (NM)')
    ylabel('North, y-position (NM)')
end

if FLAGS.hdotPlot == 1
    figure()
    plot(solutionPlot.time, solutionPlot.control(:,1)*180/pi)
    title('Heading Derivative in Degrees')
end
if FLAGS.headingPlot == 1
    figure()
    plot(solutionPlot.time, solutionPlot.state(:,3))
    title('Heading in Degrees')
end

save everything.mat
```

A.4 Three State Cost Function Code

Listing A.4: Appendix2/courseOptCost.m

```
%--------------------------------------
% BEGIN: function courseOptCost.m
%
% Variables:
%   hVec - aircraft heading vector (nm/hr)
%   rVec - vector from aircraft to radar (nm)
%   psi - angle on aircraft that faces radar in question (radian)
%   CS - cross section aircraft is displaying to radar (m) NOTE: ...
    in meters
%       in input file to make SNR calculation faster, more ...
    accurate due to
%       less conversion
%--------------------------------------
function [Mayer,Lagrange]=courseOptCost(sol)

global CONST INPUTS

tf = sol.terminal.time;
t  = sol.time;

x = sol.state(:,1);
y = sol.state(:,2);
h = sol.state(:,3);
u = sol.control.^2;
v = CONST.v;

rangeSqd = CONST.radarMaxRange.^2;
radPos = CONST.radarPos;
radStats = CONST.radarStats(:,1:6);
SNRLimit = CONST.radarStats(:,7);
```

```
radConst = CONST.radarConstant;
RCS = CONST.RCS;

Mayer = tf*10^1;

radCost = [];
for i = 1:size(rangeSqd)
    Rsqd = ((x-radPos(i,1)).^2+(y-radPos(i,2)).^2);
    CS =  RCS;
    RdB = 10.*log10((Rsqd*1852^2).^2); %convert range^4 (nm) to dB...
        (m)
    SNR = radStats(i,1)+radStats(i,2)+radStats(i,3)+radStats(i,4)+...
       CS-...
        radConst-RdB-radStats(i,5)-radStats(i,6);
    if isempty(radCost)
            radCost = 10.^((SNR-SNRLimit(i))./10);
        else
            radCost = 10.^((SNR-SNRLimit(i))./10) + radCost;
    end
end

controlCost = 10^-4.*u.^2;

Lagrange = radCost + controlCost;

%save info.mat Lagrange SNR Rsqd x y
```

A.5 Three State Differential Algebraic Equation Code

Listing A.5: Appendix2/courseOptDae.m

```
%-------------------------------------
% BEGIN: function courseOptDae.m
%-------------------------------------
function dae = courseOptDae(sol);

global CONST

t = sol.time;
x = sol.state(:,1);
y = sol.state(:,2);
h = sol.state(:,3);
u = sol.control;

xdot = CONST.v.*cos(h);
ydot = CONST.v.*sin(h);
hdot = u;

path = [];
for i = 1:size(CONST.radarPos)
    currentPath = (x-CONST.radarPos(i,1)).^2+(y-CONST.radarPos(i...
       ,2)).^2;
    path = [path currentPath];
end

dae = [xdot ydot hdot path];

%-----------------------------------
% END: function courseOptDae.m
%-----------------------------------
```

Appendix B. Five State Matlab Scripts

B.1 Five State Shell Code

Listing B.1: Appendix1/courseOptShell.m

```
% -------------------------
% Course Optimization Problem Shell Script
% -------------------------
%
% Aircraft Course Optimization Tool (ACOT)
%
% A script to aid interfacing the aircraft course optimization ...
   GPOPS
% problem designed by 2Lt Ryan Gauntt with an GUI interface. As a ...
   stand
% alone tool, this shell can also act as a method for running ...
   problems with
% the ACOT
%
% Globals:
%   INPUTS - structure containing all the necessary variables and ...
   conditions
%   for ACOT to run.
%   FLAGS - structure containing all the flags that will allow ...
   logic later on
%   in the code to run.
% Variables
%   Min_Speed - minimum ground speed of aircraft (kts)
%   Max_Speed - maximum ground speed of aircraft (kts)
%   Max_Acceleration - maximum acceleration of aircraft in flight ...
   (kts/s)
%   Max_Deceleration - maximum deceleration of aircraft in flight ...
   (kts/s)
%   Max_Load_Factor - maximum Load Factor i.e. 1g 2g 3g...
%   Max_Bank_Rate - maximum bank rate of aircraft (deg/s)
%   RCS - name of file containing radar cross section of aircraft.
%       File must currently be a .mat file with the first column ...
   as angles in
%       degrees in one degree increments and the second column the...
    RCS of
%       the aircraft at that angle
%       (m^2)
%   radar - a matrix containing all the information about each ...
   radar. This
%   will be used with the radar equation later on to produce the ...
   maximum
%   radar range. Additionally, it can be used as part of the cost ...
   function
%   as well. NOTE ON USAGE: each row corresponds to a radar. To ...
   add more
%   radars type more rows in the matrix. Variable used in matrix ...
   are
```

```
%   discussed below:
%       x_R - radar x position in local cooridnate frame (nm)
%       y_R - radar y position in local cooridnate frame (nm)
%       Ps - Power (average) transmitted from source (watts)
%       Gs - Gain of the source antenna (dB)
%       Gm - Gain of the receiving antenna (dB)
%       f - frequency of radar beam (kHz)
%       kTBn - measure of the internal noise power of the receiver...
    (dB)
%       L - loss factor (dB)
%       SNR - Signal to Noise ratio limit for detection (dB)
%       KillSNR - Signal to Noise ratio limit for definite kill [i...
   .e. SAM
%       launch] (dB) [Plane cannot enter this circle]
%   Initial Condition
%       x0 - aircraft's initial x position (nm)
%       y0 - aircraft's initial y position (nm)
%       v0_con - flag to check if initial velocity is a hard ...
   constraint
%       v0 - aircraft's initial velocity (nm/hr)
%       h0_con - flag to check if initial heading is a hard ...
   constraint
%       h0 - aircraft's initial heading (degrees) - code converts ...
   to radian
%   Final Condition
%       xf - aircraft's final x position (nm)
%       yf - aircraft's final y position (nm)
%       vf_con - flag to check if final velocity is a hard ...
   constraint
%       vf - aircraft's final velocity (nm/hr)
%       hf_con - flag to check if final heading is a hard ...
   constraint
%       hf - aircraft's final heading (degrees) - code converts to...
    radian
%   GuessPoints - The x,y coordinates that provide a reasonable ...
   guess for
%   the path the optimizer will eventually find. Recommended this ...
   be a
%   voronoi path with three points for each radar crossing (before...
    hiting
%   radars, in the middle of the two radars, and after leaving the...
    radars).
%   These are NOT waypoints that the plane has to hit, save for ...
   the initial
%   and final positions, but those are deemed necessary not by the...
    guess
%   part of the code.
%   Waypoints - Contains the row number of any guessPoint that is
%   considered a mandatory waypoint, followed by the allowable ...
   circular
%   error around that waypoint.
%   Plots - mark these as 1 if you would like the code to plot the
%   associated values.
% -------------------------

close all
clear all
clc

global INPUTS FLAGS

% Aircraft Information
Min_Speed = 200;
Max_Speed = 450;
Max_Acceleration = 50;
Max_Deceleration = 25;
Max_Load_Factor = 3; % Maximum Load Factor i.e. 1g 2g 3g...
Max_Bank_Rate = 4.5; %deg/sec
RCS_File = 'RCS55.mat';

% Radar Information
%        [x_R y_R Ps Gs Gm f kTBn L SNR KillSNR]
radars = [30 50 40000 40 40 9.9931e+006 -120 3 13 18;
          60 55 40000 40 40 9.9931e+006 -120 3 13 18;
          50 15 40000 40 40 9.9931e+006 -120 3 13 18];
          %45 10 40000 40 40 9.9931e+006 -120 3 13 50];
          %40 10 40000 40 40 9.9931e+006 -120 3 13 50;
          %80 70 40000 40 40 9.9931e+006 -120 3 13 50;
          %90 65 40000 40 40 9.9931e+006 -120 3 13 50;
          %70 20 40000 40 40 9.9931e+006 -120 3 13 50];

radar_constant = 33; %Will be a subtraction

% Initial Conditions
x0     = 0;
y0     = 50;
% Is Initial Velocity a hard constraint? If not, no need to change...
    v0 here.
v0_con = 0; % 1 = yes, 0 = no
v0     = 350;
% Is Initial Heading a hard constraint? If not, no need to change ...
   h0 here.
h0_con = 0; % 1 = yes, 0 = no
h0     = 45;

% Final Conditions
xf     = 100;
yf     = 50;
% Is Final Velocity a hard constraint? If not, no need to change ...
   v0 here.
vf_con = 0; % 1 = yes, 0 = no
vf     = 350;
% Is Initial Heading a hard constraint? If not, no need to change ...
   hf here.
```

```
hf_con = 0; % 1 = yes, 0 = no
hf     = 135;

% GuessPoints
              %[x1 y1; x2 y2;...] First and last guessPoints will ...
                  always need
              %to be x0 y0 and xf yf
guessPoints = [x0 y0;
                 30 25;
                 50 40;
                 xf yf];

% Fixed Time, Min Time, Max Time Calculation?
% If fixed time, input desired final time; otherwise leave 0.
% If max time, input desired max time; otherwise leave 0.
Final_Time = 0;

% Plots (1 = yes, 0 = no)
Plot_Guess       = 1;
Plot_Radar_SNR   = 0;
Plot_RCS         = 1;
Plot_Solution    = 1;
Plot_SolandSNR   = 0;
Plot_Colocation  = 0;
Plot_Bank_Angle  = 0;
Plot_Bank_Rate   = 0;
Plot_Heading     = 1;
Plot_Velocity    = 1;
Plot_VelDeriv    = 0;

%Convert from NAV frame to Math Frame, headings into radians
h0 = h0*(pi/180);
hf = hf*(pi/180);

C_AN = [0 1 0;
         1 0 0;
         0 0 -1];
INPUTS.C_NA = C_AN^-1;

V_N = [cos(h0); sin(h0); 0];
V_A = C_AN*V_N;
INPUTS.state.h0 = atan2(V_A(2,1),V_A(1,1));

V_N = [cos(hf); sin(hf); 0];
V_A = C_AN*V_N;
INPUTS.state.hf = atan2(V_A(2,1),V_A(1,1));

% Convert Radar input into usable numbers (freq (kHz) to ...
    wavelength (nm))
for i = 1:size(radars)
     radars(i,3) = 10*log10(radars(i,3));
    radars(i,6) = 10*log10((299710000./(radars(i,6).*1000))^2); %...
        converts from kHz frequency to m wavelenght
end

%Put all above variable into necessary structures
load(RCS_File); % NOTE: temp method, RCS must be the matrix name ...
    in file
maxRCS = 10.*log10(max(RCS(:,2))); %RCS converted to dB for use in...
     MaxRange
INPUTS.aircraft.minVelocity = Min_Speed;
INPUTS.aircraft.maxVelocity = Max_Speed;
INPUTS.aircraft.maxAcceleration = Max_Acceleration*3600; % ...
    Converting to nm/hr^2
INPUTS.aircraft.maxDeceleration = Max_Deceleration*3600; % ...
    Converting to nm/hr^2
INPUTS.aircraft.maxBankAngle = acos(1/Max_Load_Factor); % ...
    Max_Bank_Angle*pi/180;
INPUTS.aircraft.maxBankRate = Max_Bank_Rate*pi/180*3600; % ...
    Converting to radians/hr
INPUTS.aircraft.maxLF = Max_Load_Factor;
INPUTS.RCS = RCS;
for i = 1:size(radars)
    %Due to the prevelance of metric units in EM math, all range ...
        and SNR
    %calculations are done using the metric system. Even so, all ...
        units and
    %computer logic use nautical miles.
    INPUTS.radarMaxRange(i,:) = (10.^((radars(i,3)+radars(i,4)+...
        radars(i,5)+radars(i,6)+...
         maxRCS-radar_constant-radars(i,7)-radars(i,8)-radars(i,9))...
             ./10)).^(1/4)*5.39956803*10^-4;
    INPUTS.radarKillRange(i,:) = (10.^((radars(i,3)+radars(i,4)+...
        radars(i,5)+radars(i,6)+...
         maxRCS-radar_constant-radars(i,7)-radars(i,8)-(radars(i...
             ,10)))./10)).^(1/4)*5.39956803*10^-4;
end
INPUTS.radarInfo = radars;
INPUTS.radarConstant = radar_constant;
INPUTS.state.x0 = x0;
INPUTS.state.y0 = y0;
INPUTS.state.v0 = v0;
INPUTS.state.xf = xf;
INPUTS.state.yf = yf;
INPUTS.state.vf = vf;
[nuTime, nuGuess, nuControl] = courseOptGuessEnhancer(INPUTS, ...
    guessPoints);
INPUTS.guess.time = nuTime;
INPUTS.guess.state = nuGuess;
INPUTS.guess.control = nuControl;
INPUTS.finalTime = Final_Time;

plotMin = min([x0,xf,y0,yf,radars(:,1)',radars(:,2)']);
```

```
plotMax = max([x0,xf,y0,yf,radars(:,1)',radars(:,2)']);

INPUTS.plotLim.min = plotMin;
INPUTS.plotLim.max = plotMax;

if Plot_Radar_SNR == 1 || Plot_SolandSNR == 1
    disp('Calculating meshgrid values for Radar SNR Plot')
    disp('If this plot is not desired, set Plot_Radar_Cost and/or ...
       Plot_SolandRad equal to 0');
    prog = 0;
    h = waitbar(prog,sprintf('%3.0f%% Complete',prog));
    [x y] = meshgrid(plotMin:0.1:plotMax, plotMin:0.1:plotMax);
    radStats = radars(1:8);
    cost = [];
    progmax = size(radars,1)*3+1;
    for i = 1:size(radars,1)
        for j = 1:size(x,1)
            for k = 1:size(y,2)
                Rsqd(j,k) = (x(j,k)-radars(i,1))^2+(y(j,k)-radars(...
                    i,2))^2;
            end
        end
        prog = prog + 1/progmax;
        waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
        [j k] = size(Rsqd);
        S = ones(j,k);
        CS = max(RCS(:,2));
        Q = (radars(i,3)+radars(i,4)+CS+radars(i,5)+radars(i,6)...
            -...
        radar_constant-radars(i,7)-radars(i,8)).*S;
        for j = 1:size(x,1)
            for k = 1:size(y,2)
                Rdb(j,k) = 10.*log10((Rsqd(j,k)*1852^2).^2);
            end
        end
        prog = prog + 1/progmax;
        waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
        SNR = Q-Rdb;
        if isempty(cost)
            cost = SNR;
        else
            cost = cost+SNR;
        end
        prog = prog + 1/progmax;
        waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
    end
    for j = 1:size(cost,1)
        for k = 1:size(cost,2)
           if cost(j,k) > min(radars(:,10));
               cost(j,k) = min(radars(:,10));
           end
        end
    end
    prog = prog + 1/progmax;
    waitbar(prog,h,sprintf('%3.0f%% Complete',prog*100))
    close(h)
    if Plot_Radar_SNR == 1
        figure()
        axis([plotMin plotMax plotMin plotMax])
        G = surf(x,y,cost,'FaceColor','interp','FaceLighting','...
            phong');
        set(G, 'linestyle', 'none');
        title('Radar Cost Function Field')
        axis equal
        xlabel('East, x-position (NM)')
        ylabel('North, y-position (NM)')
        zlabel('Signal to Noise Ratio (dB)')
        view(2)
    end
    INPUTS.x = x;
    INPUTS.y = y;
    INPUTS.radPlots.cost = cost;

end

if Plot_Guess == 1
    figure()
    hold on
    axis([plotMin plotMax plotMin plotMax])
    for i = 1:size(radars)
        circle(radars(i,1:2),INPUTS.radarKillRange(i,:),250,'r');
        circle(radars(i,1:2),INPUTS.radarMaxRange(i),250,'g');
    end
    for j = 1:size(INPUTS.guess.state(:,1:2),1)-1
        plot([INPUTS.guess.state(j,1),INPUTS.guess.state(j+1,1)],[...
            INPUTS.guess.state(j,2),INPUTS.guess.state(j+1,2)]);
    end
    axis equal
    title('Guess Course with Radars, Exclusion Zones, and ...
        Waypoints')
    xlabel('East, x-position (NM)')
    ylabel('North, y-position (NM)')
end

if Plot_RCS == 1
    figure()
    polar(RCS(:,1).*(pi/180),RCS(:,2));
    title('Aircraft Radar Cross Section, m^2')
end
pause(0.5)

disp('Plots are behind the MATLAB window')
reply = input('Do you want continue with current guess and radar ...
    field? Y/N [Y]: ', 's');
```

```
if ~strcmp(reply,'Y') && ~strcmp(reply,'y')
    error('Run Cancelled')
end

FLAGS.v0_con = v0_con;
FLAGS.h0_con = h0_con;
FLAGS.vf_con = vf_con;
FLAGS.hf_con = hf_con;
FLAGS.solutionPlot = Plot_Solution;
FLAGS.solutionAndRadarSNR = Plot_SolandSNR;
FLAGS.colocationPlot = Plot_Colocation;
FLAGS.bankAnglePlot = Plot_Bank_Angle;
FLAGS.bankRatePlot = Plot_Bank_Rate;
FLAGS.headingPlot = Plot_Heading;
FLAGS.vPlot = Plot_Velocity;
FLAGS.vdotPlot = Plot_VelDeriv;

courseOptMain
```

B.2 Five State Guess Enhancer Code

Listing B.2: Appendix1/courseOptGuessEnhancer.m

```
function [nuTime, nuGuess, nuControl] = courseOptGuessEnhancer(...
   INPUTS, guessPoints)

%---------------------------------------
% BEGIN: function courseOptGuessEnhancer.m
%
% This script takes the main waypoints provided in the ...
   courseOptShell
% script and added in additional points between them to provide a ...
   more
% accurate initial guess to the GPOPS algorthim.
%
%---------------------------------------

maxVelocity = INPUTS.aircraft.maxVelocity;
v0 = INPUTS.state.v0;
vf = INPUTS.state.vf;
maxBankAngle = INPUTS.aircraft.maxBankAngle;

K = size(guessPoints,1)-1;
th = atan2(guessPoints(2,2)-guessPoints(1,2),guessPoints(2,1)-...
   guessPoints(1,1));
nuGuess = [guessPoints(1,1), guessPoints(1,2), v0, th, 0];
for i = 1:K
    dist = norm(guessPoints(i+1,1:2)-guessPoints(i,1:2));
    deltaTime(i) = dist/maxVelocity;
    th = atan2(guessPoints(i+1,2)-guessPoints(i,2),guessPoints(i...
       +1,1)-guessPoints(i,1));
    nuGuess = [nuGuess; nuGuess(end,1)+cos(th)*dist, nuGuess(end...
        ,2)+sin(th)*dist, maxVelocity, th, maxBankAngle];
    if nuGuess(i+1,4)-nuGuess(K)<0
        nuGuess(i+1,4) = -maxBankAngle;
    end
end
nuGuess(end,3) = vf;

totalTime = sum(deltaTime(1,:));
delTime = totalTime/K;

nuTime = 0;
nuControl = [0 0];
for i = 1:K
    nuTime(i+1) = nuTime(end)+delTime;
    nuControl = [nuControl; 0 0];
end

nuTime = nuTime';
```

B.3 Five State GPOPS Main Code

Listing B.3: Appendix1/courseOptMain.m

```
% -----------------------
% Course Optimization Problem
% -----------------------

% The problem solved here is given as follows:
% Minimize

%      t_f, J
% subject to the dynamic constraints
%      dx/dt = v*cos(u)
%      dy/dt = v*sin(u)
%      dth/dt = u
% with the boundary conditions
%      x(0) = 0, y(0) = 0, v(0) = 450
%      x(t_f) = 25, y(t_f) = 50, v(t_f) = 450
%
%  --------------------------------------------------

tic

global CONST INPUTS FLAGS

CONST.radarPos = INPUTS.radarInfo(:,1:2);
CONST.radarStats = INPUTS.radarInfo(:,3:9);
CONST.radarMaxRange = INPUTS.radarMaxRange;
CONST.radarKillRange = INPUTS.radarKillRange;
CONST.radarConstant = INPUTS.radarConstant;
CONST.vMin = INPUTS.aircraft.minVelocity;
CONST.vMax = INPUTS.aircraft.maxVelocity;
CONST.acMax = INPUTS.aircraft.maxAcceleration;
CONST.dcMax = INPUTS.aircraft.maxDeceleration;
CONST.maxBank = INPUTS.aircraft.maxBankAngle;
CONST.maxBankRate = INPUTS.aircraft.maxBankRate;
CONST.maxLF = INPUTS.aircraft.maxLF;
CONST.RCS = INPUTS.RCS;

x0 = INPUTS.state.x0;
y0 = INPUTS.state.y0;
v0 = INPUTS.state.v0;
th0 = INPUTS.state.h0;
xf = INPUTS.state.xf;
yf = INPUTS.state.yf;
vf = INPUTS.state.vf;
thf = INPUTS.state.hf;

xmin = INPUTS.plotLim.min;%-max(INPUTS.guess.state(:,1));
xmax = INPUTS.plotLim.max;%max(INPUTS.guess.state(:,1));
ymin = INPUTS.plotLim.min;%-max(INPUTS.guess.state(:,2));
ymax = INPUTS.plotLim.max;%max(INPUTS.guess.state(:,2));
vmin = CONST.vMin;
vmax = CONST.vMax;
thmin = -2*pi;
thmax = 2*pi;
phimin = -CONST.maxBank;
phimax = CONST.maxBank;
param_min = [];
param_max = [];
path_min  = CONST.radarKillRange.^2;
path_max  = xmax^2*ones(1,size(CONST.radarKillRange))';
event_min = [];
event_max = [];
duration_min = [];
duration_max = [];

iphase = 1;

limits(iphase).meshPoints = [-1 +1];
limits(iphase).nodesPerInterval = 10;
limits(iphase).time.min = [0 0];
limits(iphase).time.max = [0 10];
limits(iphase).state.min(1,:) = [x0 xmin-100 xf];
limits(iphase).state.max(1,:) = [x0 xmax+100 xf];
limits(iphase).state.min(2,:) = [y0 ymin-100 yf];
limits(iphase).state.max(2,:) = [y0 ymax+100 yf];
if FLAGS.v0_con == 1 && FLAGS.vf_con == 1
    limits(iphase).state.min(3,:) = [v0 vmin vf];
    limits(iphase).state.max(3,:) = [v0 vmax vf];
end
if FLAGS.v0_con == 0 && FLAGS.vf_con == 1
    limits(iphase).state.min(3,:) = [vmin vmin vf];
    limits(iphase).state.max(3,:) = [vmax vmax vf];
end
if FLAGS.v0_con == 1 && FLAGS.vf_con == 0
    limits(iphase).state.min(3,:) = [v0 vmin vmin];
    limits(iphase).state.max(3,:) = [v0 vmax vmax];
end
if FLAGS.v0_con == 0 && FLAGS.vf_con == 0
    limits(iphase).state.min(3,:) = [vmin vmin vmin];
    limits(iphase).state.max(3,:) = [vmax vmax vmax];
end
if FLAGS.h0_con == 1 && FLAGS.hf_con == 1
    limits(iphase).state.min(4,:) = [th0 thmin thf];
    limits(iphase).state.max(4,:) = [th0 thmax thf];
end
if FLAGS.h0_con == 0 && FLAGS.hf_con == 1
    limits(iphase).state.min(4,:) = [thmin thmin thf];
    limits(iphase).state.max(4,:) = [thmax thmax thf];
end
if FLAGS.h0_con == 1 && FLAGS.hf_con == 0
    limits(iphase).state.min(4,:) = [th0 thmin thmin];
    limits(iphase).state.max(4,:) = [th0 thmax thmax];
end
if FLAGS.h0_con == 0 && FLAGS.hf_con == 0
    limits(iphase).state.min(4,:) = [thmin thmin thmin];
    limits(iphase).state.max(4,:) = [thmax thmax thmax];
end
limits(iphase).state.min(5,:) = [phimin phimin phimin];
limits(iphase).state.max(5,:) = [phimax phimax phimax];
limits(iphase).control.min    = [-CONST.dcMax; -CONST.maxBankRate...
    ];
limits(iphase).control.max    = [CONST.acMax; CONST.maxBankRate];
limits(iphase).parameter.min  = param_min;
limits(iphase).parameter.max  = param_max;
limits(iphase).path.min       = path_min;
limits(iphase).path.max       = path_max;
limits(iphase).event.min      = event_min;
limits(iphase).event.max      = event_max;
limits(iphase).duration.min   = duration_min;
limits(iphase).duration.max   = duration_max;

guess(iphase).time(:,1)       = INPUTS.guess.time;
guess(iphase).state(:,1)      = INPUTS.guess.state(:,1);
guess(iphase).state(:,2)      = INPUTS.guess.state(:,2);
guess(iphase).state(:,3)      = INPUTS.guess.state(:,3);
guess(iphase).state(:,4)      = INPUTS.guess.state(:,4);
guess(iphase).state(:,5)      = INPUTS.guess.state(:,5);
guess(iphase).control         = INPUTS.guess.control;
guess(iphase).parameter       = [];


setup.name  = 'CourseOpt-Problem';
setup.funcs.cost = 'courseOptCost';
setup.funcs.dae = 'courseOptDae';
setup.limits = limits;
setup.guess = guess;
setup.derivatives = 'finite-difference';
setup.direction = 'increasing';
setup.autoscale = 'off';
setup.mesh.tolerance = 1e-2;
setup.mesh.iteration = 10;
setup.mesh.nodesPerInterval.min = 4;
setup.mesh.nodesPerInterval.max = 12;

[output,gpopsHistory] = gpops(setup);
solution = output.solution;
solutionPlot = output.solutionPlot;

if gpopsHistory(1,end).output.SNOPT_info == 1
    disp('Optimality conditions satisfied.')
    disp('Acceptable Course')
    disp(['Total Cost of Course: ',sprintf('%3.4f',gpopsHistory...
       (1,1).output.cost)])
else if gpopsHistory(1,end).output.SNOPT_info == 3
        disp('Optimization sucessful, but best accuracy not ...
           acheived.')
        disp('Course is usable, but optimizer had difficulty.')
        disp(['Total Cost of Course: ',sprintf('%3.4f',...
           gpopsHistory(1,1).output.cost)]);
    else if gpopsHistory(1,end).output.SNOPT_info == 41
            disp('Optimization experienced numerical difficulties....
               ')
            disp('Current point cannot be improved, though ...
               solution has been provided.')
            disp('Use of solution possible, though more analysis ...
               is required.')
            disp(['Total Cost of Course: ',sprintf('%3.4f',...
               gpopsHistory(1,1).output.cost)]);
        else
            disp('Unsucessfull Optimization.')
            disp('Ignore Output')
        end
    end
end

toc

for i = 1:size(solutionPlot.state(:,4),1)
        V_A = [cos(solutionPlot.state(i,4)); sin(solutionPlot....
           state(i,4)); 0];
        V_N = INPUTS.C_NA*V_A;
        solutionPlot.state(i,4) = acos(V_N(1,1))*180/pi;
end

if FLAGS.solutionPlot == 1 || FLAGS.solutionAndRadarSNR == 1
```

```
    figure()
    hold on
    axis([INPUTS.plotLim.min INPUTS.plotLim.max INPUTS.plotLim.min...
        INPUTS.plotLim.max])
    for i = 1:size(CONST.radarPos)
        circle(CONST.radarPos(i,:),CONST.radarKillRange(i),250,'r'...
            );
        if FLAGS.solutionAndRadarSNR == 1
            circle(CONST.radarPos(i,:),CONST.radarMaxRange(i),250,...
                'k');
        else
            circle(CONST.radarPos(i,:),CONST.radarMaxRange(i),250,...
                'g');
        end
    end
    if FLAGS.solutionAndRadarSNR == 1
        G = surf(INPUTS.x,INPUTS.y,INPUTS.radPlots.cost,'FaceColor...
            ','interp','FaceLighting','phong');
        set(G, 'linestyle', 'none');
        plot3(solutionPlot.state(:,1),solutionPlot.state(:,2),25*...
            ones(size(solutionPlot.state(:,1),1)),'k');
        view(2)
    else
        plot(solutionPlot.state(:,1),solutionPlot.state(:,2));
    end
    if FLAGS.colocationPlot == 1
    plot(solution.state(:,1),solution.state(:,2),'ro');
    end
    axis equal
    title('Optimized Aircraft Course')
    xlabel('East, x-position (NM)')
    ylabel('North, y-position (NM)')
end
if FLAGS.vPlot == 1
    figure()
    plot(solutionPlot.time,solutionPlot.state(:,3));
    title('Velocity in knots')
    xlabel('time (hrs)')
    ylabel('velocity (knots)')
end
if FLAGS.headingPlot == 1
    figure()
    plot(solutionPlot.time, solutionPlot.state(:,4))
    title('Heading in Degrees')
    xlabel('time (hrs)')
    ylabel('heading (degrees)')
end
if FLAGS.bankAnglePlot == 1
    figure()
    plot(solutionPlot.time, solutionPlot.state(:,5)*180/pi)
    title('Bank Angle in Degrees')
    xlabel('time (hrs)')
    ylabel('bank angle (degrees)')
end
if FLAGS.vdotPlot == 1
    figure()
    plot(solutionPlot.time,solutionPlot.control(:,1)/3600);
    title('Velocity Derivative in knots/sec')
    xlabel('time (hrs)')
    ylabel('acceleration (knots/sec)')
end
if FLAGS.bankRatePlot == 1
    figure()
    plot(solutionPlot.time, solutionPlot.control(:,2)*180/pi/3600)
    title('Bank Derivative in Degrees')
    xlabel('time (hrs)')
    ylabel('bank rate (degrees/sec)')
end
```

B.4 Five State Cost Function Code

Listing B.4: Appendix1/courseOptCost.m

```
%---------------------------------------
% BEGIN: function courseOptCost.m
%
% Variables:
%   hVec - aircraft heading vector (nm/hr)
%   rVec - vector from aircraft to radar (nm)
%   psi - angle on aircraft that faces radar in question (radian)
%   CS - cross section aircraft is displaying to radar (m) NOTE: ...
    in meters
%       in input file to make SNR calculation faster, more ...
    accurate due to
%       less conversion
%---------------------------------------
function [Mayer,Lagrange]=courseOptCost(sol)

global CONST INPUTS

tf = sol.terminal.time;

t = sol.time;
x = sol.state(:,1);
y = sol.state(:,2);
v = sol.state(:,3);
theta = sol.state(:,4);
phi = sol.state(:,5);
u = sol.control.^2;

rangesqd = CONST.radarMaxRange.^2;
radPos = CONST.radarPos;
radStats = CONST.radarStats(:,1:6);
SNRLimit = CONST.radarStats(:,7);
```

```
radConst = CONST.radarConstant;
RCS = CONST.RCS;

if INPUTS.finalTime ~= 0
    Mayer = exp(abs(tf-INPUTS.finalTime));
    else
    Mayer = tf*10;
end

radCost = [];

for i = 1:size(rangesqd)
    Rsqd = ((x-radPos(i,1)).^2+(y-radPos(i,2)).^2);
    hVec = [v.*cos(theta),v.*sin(theta)];
    rVec = [radPos(i,1)-x,radPos(i,2)-y];
    %psi = real(floor(acosd((hVec(:,1).*rVec(:,1)+hVec(:,2).*rVec...
       (:,2))./...
            %(sqrt(hVec(:,1).^2+hVec(:,2).^2).*sqrt(rVec(:,1).^2+...
               rVec(:,2).^2)))));
    CS =  10*log10(5);%10*log10(RCS(psi+1,2)); %converting to dB (...
       RCS in m^2)
    RdB = 10.*log10((Rsqd*1852^2).^2); %convert range^4 (nm) to dB...
        (m)
    SNR = radStats(i,1)+radStats(i,2)+radStats(i,3)+radStats(i,4)+...
       CS-...
        radConst-RdB-radStats(i,5)-radStats(i,6);
    if isempty(radCost)
            radCost = 10.^((SNR-SNRLimit(i))./10);
        else
            radCost = 10.^((SNR-SNRLimit(i))./10) + radCost;
    end
end

%gCost =  exp((1./cos(phi))-1)-1;
controlCost = sum([10^-15 0; 0 10^-8]*[transpose(u(:,1))*u(:,1);...
    transpose(u(:,2))*u(:,2)]);

Lagrange = radCost + controlCost;
```

B.5 Five State Differential Algebraic Equation Code

Listing B.5: Appendix1/courseOptDae.m

```
%-------------------------------------
% BEGIN: function courseOptDae.m
%-------------------------------------
function dae = courseOptDae(sol)

global CONST

t = sol.time;
x = sol.state(:,1);
```

```
y = sol.state(:,2);
v = sol.state(:,3);
h = sol.state(:,4);
b = sol.state(:,5);
u = sol.control;

LF = 1./cos(b);

xdot = v.*cos(h);
ydot = v.*sin(h);
vdot = u(:,1);
hdot = LF.*32.2/6079*3600^2.*sin(b)./v; % Load Factor to nm/hr^2
bdot = u(:,2);

pathCon = [];
for i = 1:size(CONST.radarPos)
    currentPathCon = (x-CONST.radarPos(i,1)).^2+(y-CONST.radarPos(...
       i,2)).^2;
    pathCon = [pathCon currentPathCon];
end

dae = [xdot ydot vdot hdot bdot pathCon];

%-----------------------------------
% END: function courseOptDae.m
%-----------------------------------
```

Bibliography

1. Herbert, J. M., "Air Vehicle Path Planning," 2001.
2. Sharma, R., "Funadamentals of Tactical Missiles," 2007.
3. Novy, M. C., "Air Vehicle Optimal Trajectories for Minimization of Radar Exposure," 2001.
4. Cottrill, G. C. and Harmon, F. G., "Hybrid Gauss Pseduospectrol and Generalized Polynomial Chaos Algorithm to Solve Stochastic Trajectory Optimization Problems," .
5. Timothy R. Jorris, R. G. C., "2-D Trajectory Optimization Satisfying Waypoints and No-Fly Constraints," 2007.
6. Abbot, L., Stillings, C., Phillips, C., and Knowlan, G., "Annex A, Section 3 of Risk Management Plan for the Fleeting Target Technology Demontrator," 2007.
7. Jorris, T., "Dynamic Optimization and GPOPS Training," PowerPoint training slides.
8. Rao, A. V., "User's Manual for GPOPS Version 4.x: A MATLAB(R) Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Pseudospectral Methods," 2011.
9. Gill, P. E., Murray, W., and Saunders, M. A., "User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming," 2008.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 22-03-2012	**2. REPORT TYPE** Masters Thesis	**3. DATES COVERED** *(From - To)* March 2010 - March 2012

4. TITLE AND SUBTITLE Aircraft Course Optimization Tool Using GPOPS MATLAB Code	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Ryan Gauntt, 2Lt	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 641 WPAFB, OH 45433-8865	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GSE/ENV/12-M03
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Lt Col Paul Blue - paul.blue@stratcom.mil USSTRATCOM JFCC GSI/J57 901 SAC BLVD, NE 68113 Phone: Comm (402) 294-1497 DSN 271-1497 FAX: Comm (402) 232-6641 DSN 2726641	**10. SPONSOR/MONITOR'S ACRONYM(S)** GSI/J57 **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United states.

14. ABSTRACT
Aircraft course planning between two points in a clear, no threat environment is easy and straightforward. However, the addition of various threats can greatly increase the difficulty and complexity of course planning. Placing new waypoints along the edge of each threat, mostly skirting the dangerous environment, may not prove too difficult, but such courses are far from optimal. Given aircraft, environment, and time constraints it is likely a much more optimal path exists between any given start and end points. This research focuses on determining the feasibility of using the General Pseudospectral Optimization Software program files written for the MATLAB software package to take a given path, optimize it for the environment, and output a flyable, optimized course that can be used for more detailed mission planing. The results showed creating such a code was feasible. GPOPS can handle a simple version of what could be a very complex optimization problem. Two different versions of the final code show the successful optimization of the problem when the model is kept simple, and the failures GPOPS experiences when the problem becomes too complex.

15. SUBJECT TERMS
aircraft, course, optimization, radar, GPOPS, SNOPT, MATLAB

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. David Jacques
U	U	U	UU	103	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x3329